НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів»

Виконав:
студент IV курсу, групи КП-61
Герасимов Артем Сергійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н, доцент,
Дідковська Марина Віталіївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Герасимову Артему Сергійовичу

1. Тема проєкту «Веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів», керівник проєкту Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету від «___»_____ 2020 р. №_____

2. Термін подання студентом проєкту «___» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих програмних рішень;
   - розроблення web-системи для автоматизованого виявлення та оброблення скарг Інтернет-користувачів;
   - опис розроблених алгоритмів та підпрограм;
   - аналіз розробленої web-системи.

5. Перелік обов'язкового графічного матеріалу:
   - авторизація користувача (креслення);
   - синхронізація даних (креслення);

- схема архітектури web-системи (плакат);
- графічний компонент відображення статистики (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	28.11.2019	
3.	Розроблення структури web-ресурсу	15.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	20.02.2020	
7.	Програмна реалізація web-ресурсу	10.03.2020	
8.	Тестування web-ресурсу	17.03.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	11.04.2020	
11.	Підготовка графічної частини дипломного проєкту	21.04.2020	
12.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Артем ГЕРАСИМОВ

Керівник проєкту Тетяна ЗАБОЛОТНЯ

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню веб-додатка для автоматизованого виявлення та оброблення скарг інтренет-користувачів. Тема роботи обумовленна відсутністю аналогів системи для автоматизованого моніторингу за скаргами на заклади та сервіси та критичною необхідністю бізнесу в їх наявності.

Програмний комплекс був створений базуючись на функціональних та бізнес вимогах, продиктованих ринком сучасної сфери послуг. Було виділено границі проєкту та мінімально життєздатний продукт для забезпечення таких вимог.

Розроблена система є комплексом із клієнтських та серверних застосунків із єдиним персистентним сховищем даних. Кожен компонент відповідає сучасним тенденціям розробки програмного забезпечення, а стилістика написання програмного коду слідує базовим парадигмам програмування. Адміністраторська панель візуалізується за допомогою сучасних динамічних веб-компонентів, сервер реалізує усі принципи RESTful API, а модулі виявлення та оброблення тексту надають абстрактні інтерфейси над найсучаснішими технологіями NLP.

У даному дипломному проєкті розроблено: модуль збору скарг інтернет-користувачів, модуль аналізу текстових фрагментів, закритий RESTful API для надання високорівневих інтефейсів взаємодії із системою та клієнтська частина у вигляді стилізованого веб-додатку із адаптивним дизайном та реактивними компонентами. Агентами комунікації між компонентами системи стали протоколи HTTP та WebSockets.

ABSTRACT

The presented diploma project is dedicated to implementation of web application for automated detection and processing of internet users' complaints. The topic of thesis was stipulated by the absence of analogs in automated monitoring systems for establishments and services, as well as by the business's crucial demand.

This software complex was developed based on functional and business requirements that were dictated by the contemporary service industries. The boundaries of the project and the minimal viable product were established beforehand to satisfy those requirements.

The developed software is a complex that comprises client and server applications with the single persistent storage. Each component follows modern tendencies of software development and the style of code adhere to the paradigms of software engineering. The administration panel is visualized with the help of novel dynamic web components, server-side meets the principles of RESTful API, and the modules of complaints detection and processing provide abstract interfaces over bleeding-edge NLP technologies.

The following modules were developed within the diploma project: module of complaints collection, module of text processing, private RESTful API that grants high level interfaces for interaction with the system and the client module in the form of stylized web application with adaptive design and reactive components. The agents of communication are HTTP and WebSockets protocols.

АННОТАЦИЯ

Данный дипломный проект посвящен созданию веб-приложения для автоматизированного определения и обработки жалоб интернет-пользователей. Тема работы обусловлена отсутствием аналогов системы для автоматизированного мониторинга жалоб, которые поступают на заведения и сервисы, а также, критичной потребностью бизнеса в их доступности.

Программный комплекс был создан на базе функциональный и бизнес требований, продиктованных современным рынком сферы услуг. Были очерчены границы проекта и минимально жизнеспособный продукт для обеспечения таких требований.

Разработанная система является комплексом из клиентский и серверных приложений с единым персистентным хранилищем данных. Каждый компонент отвечает современным тенденциям разработки програмного обеспечения, а стилистика написания програмного кода следует базовым парадигмам программирования. Администраторская панель визуализируется при помощи современных динамических веб-компонентов, сервер реализует все принципы RESTful API, а модули определения и обработки жалоб предоставляют абстрактне интерфейсы над передовими технологиями NLP.

В данном дипломном проекте разработано: модуль сбора жалоб интернет-пользователей, модуль анали за текстових фрагментов, закрытый RESTful API для предоставления высокоуровневых интерфейсов взаимодействия с системой та клиентской частью в виде стилизованного веб-приложения с адаптивным дизайном и реактивными компонентами. Агентами коммуникации между компонентами системы стали протоколы HTTP та WebSockets.

ДП.045440-01-90. Веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-додаток для	5	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Технічне завдання		
ДП.045440-03-81	Веб-додаток для	50	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-додаток для	4	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-додаток для	10	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-додаток для	1	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Структура бази даних.		
	ERD-діаграма		
ДП.045440-07-99	Веб-додаток для	1	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Функціональність		
	програмних засобів.		
	Діаграма прецидентів		
ДП.045440-08-98	Веб-додаток для	1	
	автоматизованого		
	виявлення та оброблення		
	скарг		
	Інтернет-користувачів.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗОВАННОГО ВИЯВЛЕННЯ ТА ОБРОБЛЕННЯ СКАРГ ІНТЕРНЕТ-КОРИСТУВАЧІВ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ГЕРАСИМОВ

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: веб-додаток для автоматизованого виявлення та оброблення скарг інтернет-користувачів.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості допоміжного інструменту для виявлення та оброблення скарг Інтернет-користувачі.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмна система повинна забезпечувати такі основні функції:

1. Можливість реєстрації та авторизації компанії у системі.
2. Створення циклів виявлення та оброблення скарг та специфікація додаткової інформації щодо циклу.
3. Перегляд статусу виконання циклу у реальному часі.
4. Перегляд результатів циклу у вигляді діаграми типів скарг, списку всіх наявних скарг та відсоткового складу категорій скарг.
5. Перегляд архіву виконаних циклів.
6. Оновлення базових даних про компанію.
7. Параметризація періодів виконання циклів.

Розроблення клієнтської частини необхідно виконати на базі сучасного фреймворку із динамічною роботою із DOM-деревом. Серверну частину реалізувати за допомогою фреймворку, що надає високорівневі функції для створення RESTful API. Розроблення алгоритмів класифікації виконати за допомогою мови програмування Python на базі типових бібліотек.

Додаткові вимоги:

1. Можливість зупинки циклу виявлення та оброблення скарг.
2. Можливість динамічного оновлення веб-сторінок.
3. Імплементація інтерфейсу користувача із дотриманням методологій Material design.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1. Програма та методика тестування.
2. Керівництво користувача.
3. Креслення:
4. Пояснювальна записка.

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту 16.11.2019
Розроблення та узгодження технічного завдання..... 09.12.2019
Підготовка матеріалів першого розділу дипломного проєкту 30.12.2019
Розроблення алгоритмів класифікації відгуків 16.01.2019
Підготовка матеріалів другого розділу дипломного проєкту..... 10.02.2019
Програмна реалізація та тестування програмної системи 20.02.2020
Підготовка третього розділу дипломного проєкту..... 10.03.2020
Підготовка четвертого розділу дипломного проєкту..... 11.04.2020

Підготовка графічної частини дипломного проєкту 19.05.2020

Оформлення документації дипломного проєкту 26.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ТА ОБРОБЛЕННЯ СКАРГ ІНТЕРНЕТ-КОРИСТУВАЧІВ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ГЕРАСИМОВ

2020

ЗМІСТ

ВСТУП

Оптимізація процесів збору та аналізу скарг інтернет-користувачів є актуальною проблемою для закладів та сервісів у сфері послуг. Конкуренція в розробці динамічних інструментів аналізу відгуків стає підґрунтям для технологічних пошуків для великих та гнучких корпорацій швидкого харчування, соціальних мереж та державних установ.

По-перше, акумуляція текстових фрагментів інтернет-відгуків є витратною у часі й ресурсах задачею, оскільки інформація дисимільована на різних, взаємо непов’язаних сервісах із відмінними рейтинговими системами, фільтрами для сортування тощо. По-друге, навіть за наявності системи збору та динамічного відстеження відгуків постає проблема їх автоматизованого аналізу. Сьогодні таку обробку виконують мануально працівники відділів досліджень. Створення комплексу, що включає як збір, так і аналіз, у формі автоматизованої класифікації, вимагає просунутого досвіду команди розробників у імплементації алгоритмів обробки природніх мов та машинного навчання. Очевидно, що процес створення такої системи вимагатиме суттєві фінансові та часові витрати.

Слід зазначити, що розроблення описаної вище системи не є гарантією комерційного успіху та фактом усунення проблем бізнесу, оскільки вимагає мануального адміністрування та аналізу отриманих результатів.

Сучасні практики створення гнучких веб-сервісів дозволяють створювати якісні адміністративні інструменти. Дійсно, зазначені раніше компанії зможуть ефективно інтегрувати алгоритмічну систему в існуючі маркетингові стратегії лише за наявності інтуїтивного інтерфейсу користувача, що слідує класичним методикам розроблення ПЗ. Навчання існуючого персоналу базовим інструкціям по використанню веб-додатку повинна бути зведена до мінімуму.

Доречно проілюструвати типовий випадок використання системи у бізнес умовах. Нехай дано мережу закладів швидкого харчування як головний об’єкт дослідження. У відокремленому закладі мережі відвідувачі почали скаржитись на відсутність рушників у вбиральнях. На наступний день, відвідувачі обирають конкуруючий заклад. Адміністратор проблемної інституції звертається до адміністраторської панелі розробленого комплексу. На візуалізованій карті проблем, зібраних із онлайн-джерел відгуків, він бачить найгострішу проблему, описану вище. Після колегіальної наради приймається рішення додати по одному рушнику в кожну вбиральню. Після специфікації адміністратором “кроків до вирішення” у системі запускається наступний етап збору відгуків. Високою є ймовірність розрішення проблеми, індикатором чого буде оновлена карта проблем після вказаного періоду.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API, Application Programming Interface – набір функцій та процедур, що дозволяють створювати додатки, які отримують доступ до функцій або даних операційної системи, програми чи іншої послуги.

RESTful API – це інтерфейс прикладної програми, який використовує HTTP-запити типів GET, PUT, POST та DELETE для маніпуляції даними сервісу.

БД – база даних.

JSON, JavaScript Object Notation – це відкритий стандарт формату файлу та формат обміну даними, який використовує текст для зберігання та передачі об’єктів даних, що складаються з пар ключ-значення.

Unit-тестування – це рівень тестування програмного забезпечення, на якому тестуються окремі блоки/компоненти програмного забезпечення.

End-to-end-тестування – це техніка, що використовується для перевірки того, чи потік програми від початку до кінця веде себе так, як очікувалося.

Health check – перевірка функціонування сервісу.

ПЗ – програмне забезпечення.

Desktop-застосунок – програмний продукт, що функціонує на базі настільних операційних систем.

DOM, Document Object Modal – являє собою міжплатформенний і незалежний від мови інтерфейс, що розглядає XML або HTML документ як структуру дерева, де кожен вузол є об'єктом, що представляє частину документа.

Node.js – це відкрите, кросплатформене середовище виконання JavaScript, яке виконує код JavaScript поза веб-браузером.

Фреймворк – являє собою абстракцію, у якій програмне забезпечення, що забезпечує загальну функціональність, може вибірково

змінюватися додатковим кодом, написаним користувачем, забезпечуючи таким чином програмне забезпечення, яке стосується додатків.

NLP, Natural Language Processing – це підрозділ лінгвістики, інформатики та штучного інтелекту, пов'язаний із взаємодією між комп'ютерами та людськими (природними) мовами, зокрема, як програмувати комп'ютери для обробки та аналізу великої кількості природних мовних даних.

GP – Google Places.

CRUD-операції – функції створення, видалення, оновлення та зчитування.

1. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Обґрунтування вибору критеріїв оцінювання

Для створення веб-додатку для автоматизованого виявлення та оброблення скарг інтернет-користувачів було висунуто такі вимоги:

1. Усі скарги повинні бути розподілені на категорії і підкатегорії. Це необхідно зробити для більш зручного пошуку необхідного для користувача набору проблем.
2. Створення власних категорій проблем та забезпечення гнучкості системи.
3. У системі необхідний модуль Push-оповіщень про виявлення нових проблем або резолюцій тих, що активно відслідковували.
4. У системі повинна бути можливість специфікації періоду відстеження скарг. Можливість динамічної модифікації періоду.
5. Система повинна працювати при виході із ладу будь-якого компоненту. Наявна можливість збирання даних із різних джерел для запобігання перебоїв через недоступність одного з них.
6. Система повинна надати адміністратору можливість переглядати відгуки в первинному вигляді для вилучення неточностей класифікації та точковому виявленні проблем.
7. У системі повинна бути як користувацька панель для кінцевого користувача, так і адміністраторська панель для контролю стану бази даних та модулів збору, аналізу, збереження інформації.
8. Компоненти графічного інтерфейсу повинні мати ієрархічну структуру із наявністю незалежних функціональних компонентів.
9. Система повинна мати такі форми авторизації, реєстрації, відображення проблем та списку проблем у кожній категорії,

форма відображення проблем по категоріям, форма встановлення та модифікації періодів збору даних.

10. Наявність англомовної локалізації, оскільки система орієнтована на міжнародний ринок надання послуг та сервісів.

1.2. Огляд існуючих сервісів для аналіз скарг користувачів

1.2.1. Zendesk

Zendesk (zendesk.com) – це сервіс, що інтегрується у більшість сучасних систем та дозволяє користувачам залишати відгуки та спілкуватися із адміністраторами сервісів у режимі реального часу. Даний сервіс є прикладом розробки інтерфейсу веб-панелі, оскільки надає смогу не лише відслідковувати скарги, але й візуалізувати статистичні дані.

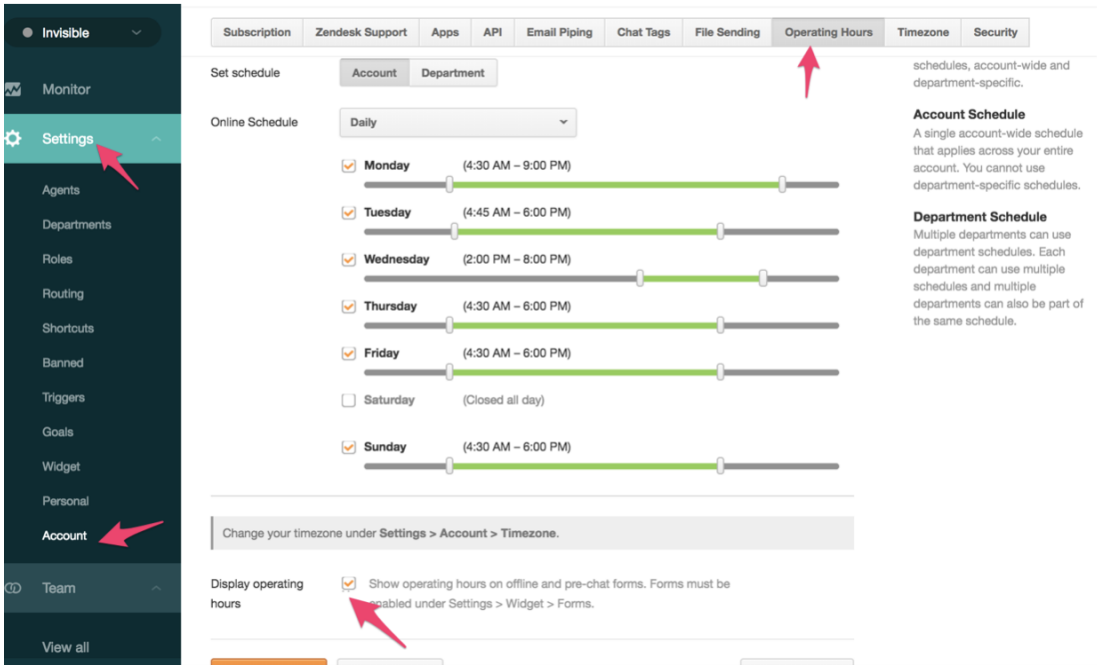


Рис. 1.1. Головна сторінка веб-сервісу Zendesk

Переваги:

1. Інтегровність у всі види сучасних систем.
2. Зручний веб-інтерфейс для адміністрування.
3. Інструменти візуалізації скарг.

Недоліки:

1. Висока ціна. Близько 109$ на місяць для одного користувача.
2. Відсутність чату в реальному часі.
3. Відсутність можливості автоматизованого збору відгуків із популярних джерел, на кшталт Google Places.
4. Відсутність алгоритмів класифікації та категоризації.

1.2.2. Intercom

Intercom (intercom.com) – аналог Zendesk з певними суттєвими відмінностями. Розрахований для компаній, що орієнтуються на найсучасніші технології чат-листування, а не на зручність веб-інтерфейсу. Intercom розрахований на менші підприємства, що орієнтуються на чат у реальному часі, проте відсутність спеціалізованих статистичних інструментів не є суттєвою.

Переваги:

1. Можливість чат-листування у режимі реального часу.
2. Легка інтеграція із більшістю сучасних інтерфейсів.

Недоліки:

1. Відсутні статистичні інструменти оброблення даних.
2. Застаріла бібліотека для Android, IOS та React Native платформ для мобільної розробки.
3. Висока ціна для інструменту із обмеженим адміністраторським функціоналом.

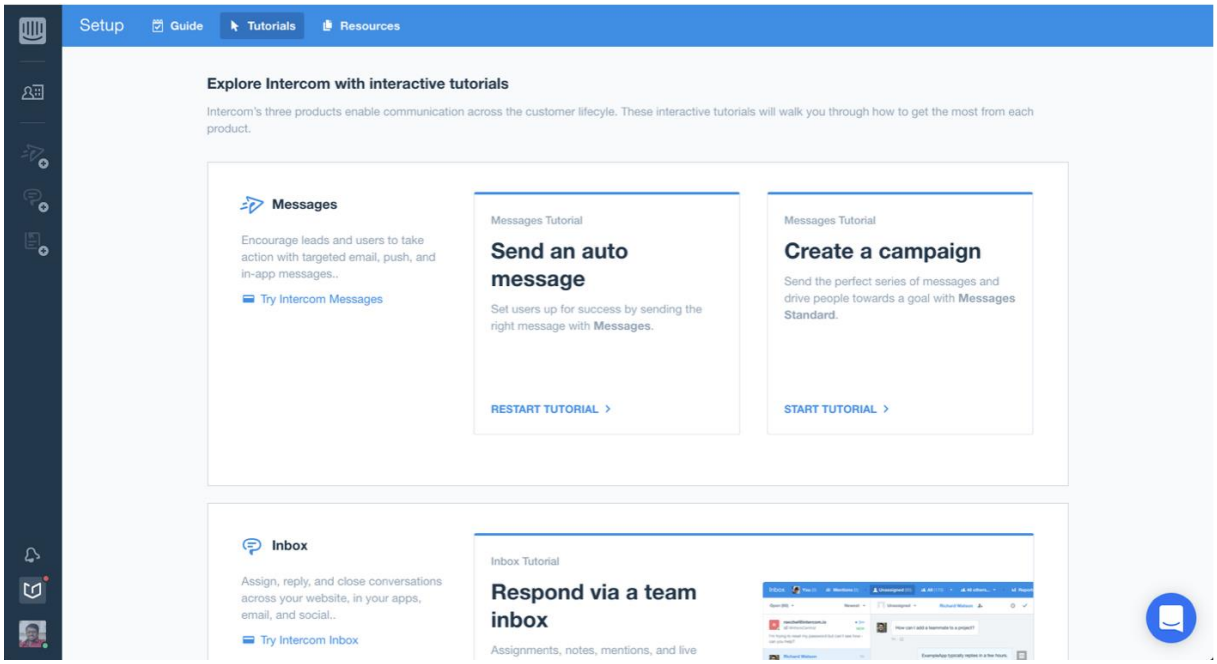


Рис. 1.2. Головна сторінка веб-додатку Intercom

1.2.3. Google Places

Google Places – це сервіс, що є складовою системою еко-системи Google. Він надає можливість користувачам залишати відгуки про місця, які вони відвідали, виставляти рейтингові оцінки, додавати фото тощо.

Даний сервіс є найбільшим агрегатором відгуків, проте, не надає жодного бізнес-інструментарію для автоматизованого виявлення та аналізу скарг. Звідси слідує, що Google Places може бути корисним лише для індивідуального використання пересічним користувачем та ніяк не може бути багатофункціональним інструментом промислового масштабу. Проте, Google надає API для отримання даних про відгуки користувачів, що робить Places API ідеальним джерелов для агрегації текстових фрагментів для сторонніх сервісів. Якщо користувач API не є власником бізнесу, то він має доступ лише до п’яти найпопулярніших скарг, що виключає використання сервісу напряму. Для вирішення цієї проблеми було прийнято рішення використати сервіс Wextractor для отримання необмеженої кількості відгуків посередньо через Google Places API. Зазначимо ще раз переваги й недоліки сервісу Google Places. Переваги:

1. Наявність API, що може бути використаний для побудови більш грунчких та аналітичних систем.

Недоліки:

1. Відсутність автоматичного виокремлення скарг з-поміж відгуків.
2. Відсутність категоризації відгуків.

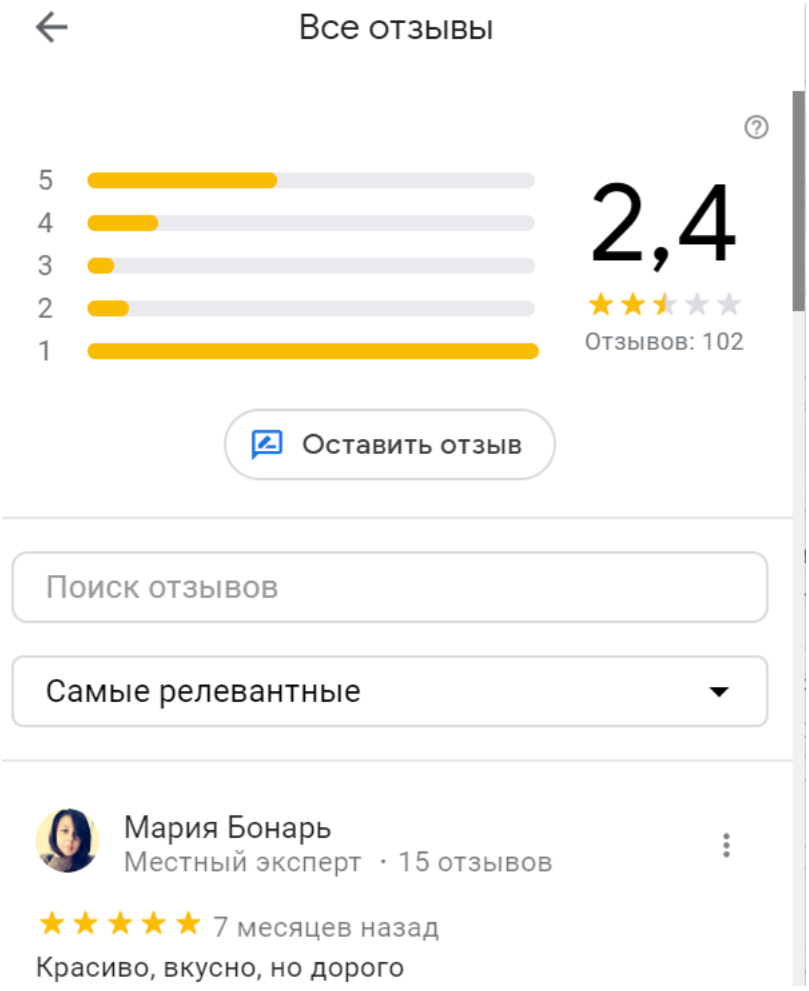


Рис. 1.3. Приклад перегляду відгуків у Google Places

1.3. Результати аналізу

Виходячи з огляду існуючих рішень, можна зазначити, що жодне з досліджуваних рішень не задовольняє більшості висунутих вимог.

По-перше, запропоновані програмні засоби відповідають вимозі моніторингу онлайн-скарг. Але щоб повністю відповідати усім потребам, система повинна мати можливість розподіляти скарги за категоріями, чого

жодна із запропонованих систем не пропонує. Також, усі ці системи є спеціалізованими додатками для спілкування й вирішення проблем користувачів у режимі консалтингу, проте, за вимогами до розробленого ПЗ, увесь збір та аналіз має бути автоматизованим.

По-друге, наведені вище системи не є достовірно безпечними у розрізі конфіденційності наданих користувачами та адміністраторів даних. Дійсно, Intercom та Zendesk є інтегровними та передають функціональний контроль розробникам. Так, існує теоретична можливість перехоплення даних про недоліки закладів або стратегій їх резолюції. У той час, розроблюване ПЗ є незалежним продуктом, що віддається підприємству у користування як окремий завершений продукт. Звідси, він може функціонувати під локальними захищеними мережами й повністю контролюватися системними адміністраторами клієнтів даного продукту.

Порівняльна характеристика аналізованих систем подана у табл. 1.1.

Таблиця 1.1

Порівняння конкуруючих продуктів

	Ціна, $/місяць	Автоматизація	Зворотній зв'язок	Чат	Статистика
Zendesk	109	Відсутні	–	+	+
Intercom	159	Відсутні	–	+	–
GP	0	Відсутні	–	–	–

Легко побачити, що Zendesk точніше наслідує описану в даній роботі систему та частково повторює алгоритмічні інструменти на стороні веб-панелі. У той час, Intercom є засобом, що вирішує схожі проблеми бізнесу, проте, абсолютно іншими шляхами, що не є автоматизованими та ефективними.

Отже, проведений аналіз дозволяє зробити висновок про необхідність створення автоматизованої системи для збору та аналізу скарг інтернет-користувачів, що вирішить кореневу проблему гнучкої адаптації під сучасні вимоги клієнтів.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір мови програмування

Вимоги до мови програмування, що випливають з постановки задачі:

1. Велика кількість сторонніх модулів та активної спільноти.
2. Наявність фреймворків для створення RESTful API.
3. Асинхронність у обробці.

2.1.1. JavaScript

Javascript – об’єктно-орієнтовна мова програмування, що реалізує стандарт ECMAScript. Скриптова мова програмування, що була розроблена для створення динамічних веб-сторінок, проте, може бути використана для розроблення програмного забезпечення для більшості сучасних платформ: Desktop-застосунки, мобільні додатки, веб-сайти тощо [1]. Мова є однопотоковою та підтримує асинхронність виконання функцій. Саме ці особливості диктують тенденцію переходу від багатопотокових мов програмування на серверній частині. Дійсно, із використанням JavaScript зникає необхідність у обробці запитів клієнтів у окремих ресурсномістких потоках. Більш того, динамічна типізація, простота синтаксису мови та наявність активної спільноти розробників і open-source бібліотек робить Javascript підходящою для швидкого прототипування програмних продуктів. Проте можна виділити наступні недоліки Javascript як мови програмування:

1. Помилки кодування браузера та плагінів
2. Схильність до проблем із XSS (cross-site scripting)
3. Схильність до неявних помилок, пов’язаних із динамічною типізацією.
4. Мала ефективність у виконанні задач, пов’язаних із процесорною обробкою.

2.1.2. TypeScript

TypeScript – мова програмування, що є розширенням мови JavaScript. Працює на основі інтерпретатора, що транскомпілює код в Javascript та є зворотньо сумісним з ним [2]. TypeScript підтримує файли визначення, які можуть містити інформацію про типи існуючих бібліотек JavaScript, подібно до файлів заголовків C ++, можуть описувати структуру існуючих об'єктних файлів. Це дає можливість іншим програмам використовувати значення, визначені у файлах, так, як якщо б вони були статично набраними сутностями TypeScript. Існують сторонні файли заголовків для популярних бібліотек, таких як jQuery, MongoDb і D3.js. Також доступні заголовки TypeScript для основних модулів Node.js, що дозволяє розробляти програми Node.js у TypeScript. Це є суттєвою перевагою для вибору даної мови програмування для розробки одночасно клієнтської та серверної частин системи, оскільки виникає зручність у повторному використанні інтерфейсів для загальних сутностей.

Як висновок, для розробки даної системи була обрана мова TypeScript, оскільки вона вміщає усі переваги JavaScript та надає ар

Надалі, хітектурні переваги для масштабовності та гнучкості розробленої системи. перейдемо до аналізу можливих фреймворків для стороні клієнту на серверу, що мають підтримку TypeScript.

2.2. Вибір технології для розроблення серверної частини системи

2.2.1. Express.js

Express.js – бібліотека JavaScript, яка використовується для розробки ефективних, швидких та масштабованих веб-додатків, використовуючи ще одну бібліотеку JavaScript під назвою Node.js [3].

Для використання необхідно імпортувати бібліотеку Express.js у файл Node.js, оскільки це надає багато готових до використання функцій

для написання частини маршрутизації веб-програми. Без використання Express.js знадобиться багато програмного коду та зусиль для того, щоб розробити ефективний і правильний компонент маршрутизації для веб-сторінок / API.

На жаль, Express.js нативно не підтримує TypeScript, тому для досягнення такої комбінації технологій необхідні додаткові конфігурації, але TypeScript буде більше імітуватися, ніж виконувати своє призначення. Тому перейдемо до наступної опції, що долає цей недолік.

2.2.2. NestJS

NestJS – це фреймворк для створення ефективних масштабованих додатків на сервері Node.js. Він використовує прогресивний JavaScript, створений на TypeScript із повною його підтримкою і поєднує елементи об'єктно-орієнтованого функціонального та функціонально-реактивного програмування [4]. NestJS побудований на основі ExpressJS, тому містить усі його переваги. Даний фреймворк маю широку бібліотеку класів для оброблення HTTP-запитів, формування відповідей, валідації даних користувача та доступу до бази даних. Також наявні декоратори, що синтаксично допомагають приховати багато повторюваного коду та пришвидшити швидкість розробки. Висока структурованість коду досягається за допомогою розділення коду на модулі, провайдери та контролери. Розділивши програмний код на модулі можна зручно адмініструвати версіонність API та розділити частини сервера для використання різними клієнтами.

Як висновок, для розробки серверної частини даної системи був обраний NestJS через повну сумісність із обраною мовою програмування та широким влаштованим функціоналом для створення REST-ful API.

2.3. Вибір технології для розроблення клієнтської частини системи

2.3.1. AngularJS

AngularJS – це веб-фреймворк з відкритим вихідним кодом, заснований на JavaScript, який підтримується Google, а також спільнотою осіб та корпорацій для вирішення багатьох проблем, що виникають при розробці програм на клієнтській сторінці [5]. Він спрямований на спрощення як розробки, так і тестування таких додатків, забезпечуючи структуру для клієнтської архітектури модель-перегляд-контролер (MVC) та модель-перегляд-перегляд-модель (MVVM), а також компоненти, що часто використовуються в багатьох Інтернет-додатках.

AngularJS працює, попередньо прочитавши сторінку мови розмітки гіпертексту (HTML), на якій вбудовані додаткові спеціальні атрибути HTML. Фреймворк інтерпретує ці атрибути як директиви для прив’язки вхідних або вихідних частин сторінки до моделі, яка представлена стандартними змінними JavaScript. Значення цих змінних JavaScript можна встановити вручну в коді або отримати зі статичних або динамічних ресурсів JSON.

AngularJS побудований на припущені, що декларативне програмування слід використовувати для створення інтерфейсів користувача та підключення програмних компонентів, тоді як імперативне програмування краще підходить для визначення бізнес логіки програми. AngularJS адаптує та розширює традиційний HTML для представлення динамічного контенту за допомогою двостороннього зв’язування даних, що дозволяє здійснювати автоматичну синхронізацію моделей та інтерфейсу користувача. Як результат, AngularJS концентрує увагу на явній маніпуляції з об'єктною моделлю документа (DOM) з метою поліпшення тестування та ефективності.

2.3.2. VueJS

Vue.js – це фреймворк на основі JavaScript для моделювання інтерфейсу користувача та односторінкових програм з відкритим кодом [6]. Vue.js дозволяє розширювати HTML за допомогою атрибутів, які називаються директивами. Директиви розширюють функціональні можливості для HTML сторінок і надходять як вбудовані або визначені користувачем директиви. Серед суттєвих переваг Vue.js можна виокремити такі:

1. Шаблони. Vue використовує синтаксис шаблонів на основі HTML, який дозволяє прив'язувати наданий DOM до базових даних екземпляра Vue. Усі шаблони Vue є дійсним HTML, який можна проаналізувати за допомогою веб-переглядачів та HTML-аналізаторів.
2. Реактивність. Vue має систему реактивності, яка використовує звичайні об’єкти JavaScript та оптимізує повторну візуалізацію. Кожен компонент відслідковує свої реактивні залежності під час візуалізації, тому система точно знає, коли потрібно відредагувати, та які компоненти повторно рендерувати.

2.3.3. ReactJS

ReactJS – це бібліотека JavaScript для побудови користувальницьких інтерфейсів [7]. Він підтримується Facebook та спільнотою окремих розробників та компаній.

React можна використовувати як базу при розробці односторінкових або мобільних додатків. Однак React стосується лише надання даних у DOM, тому створення React додатків зазвичай вимагає використання додаткових бібліотек для управління станом та маршрутизації (Redux та React Router).

Код React проекту складається з об'єктів, що називаються компонентами. Компоненти можуть бути передані певному елементу

в DOM за допомогою бібліотеки React DOM. Під час візуалізації компонента можна передати додаткові властивості для рендерингу. Ще одна примітна особливість – використання віртуальної моделі об'єкта документа або віртуальної DOM. React створює кеш даних, обчислює отримані відмінності, а потім ефективно оновлює відображений DOM. Це дозволяє програмісту писати код так, ніби вся сторінка відображається при кожній зміні, в той час як бібліотеки React надають лише підкомпоненти, які фактично змінюються.

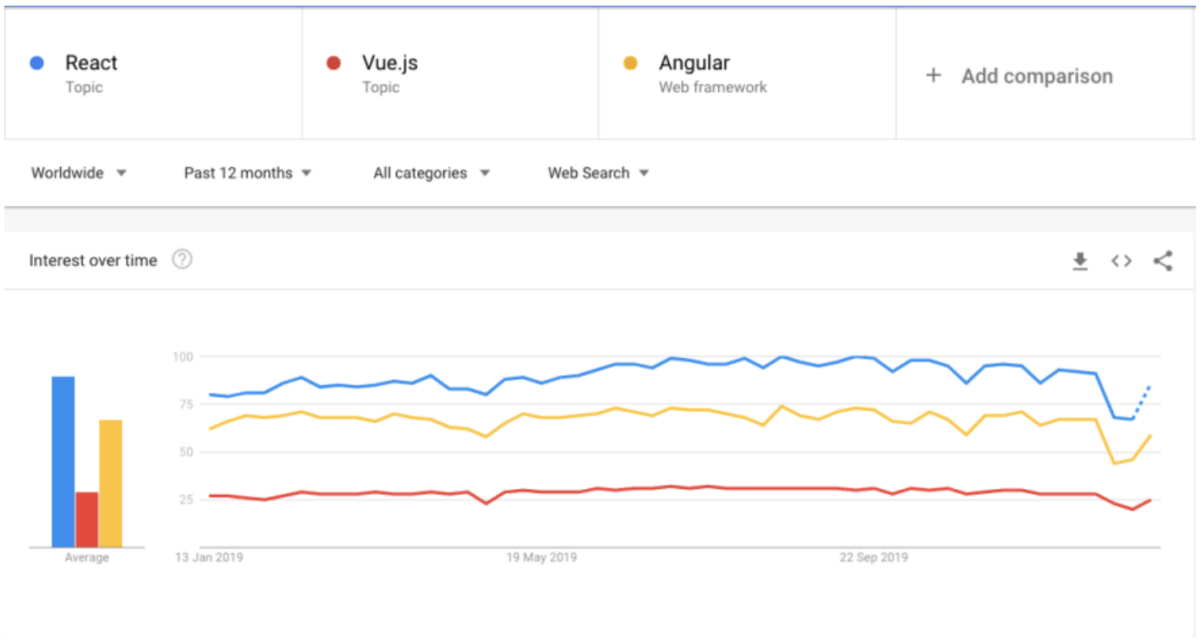


Рис. 2.1. Порівняння популярності сучасних веб-фреймворків

Саме ReactJS був обраний як базова технологія створення клієнтської частини даної системи, оскільки веб-інтерфейс, у зв’язку із вимогами до ПЗ, містить багато динамічних елементів, що потребують максимально оптимізованих інструментів для рендерингу. Більш того, за статистикою, React є найпопулярнішим інструментом для створення клієнтських застосунків (рис. 2.1). Із цього слідує, що у відкритому доступі наявна найбільша кількість сторонніх доповнень, компонентів і бібліотек, що прискорюють розробку та підвищують якість кінцевого продукту.

Більш того, більша популярність технології означає більшу кількість робочих ресурсів готових до виконання відповідної роботи. Також, компонентний підхід дозволяє масштабувати систему для мобільних пристроїв для стаціонарних комп'ютерів через повторне використання веб-компонентів у рамках ReactNative (для мобільних девайсів) або ProtonNative (для Desktop).

2.4. Вибір моделі для класифікації тексту

Для створення класифікатора, необхідно обрати готову модель, яка далі буде використана у якості Black box для розмічених даних. Припустимо, що тестування існуючих моделей проводилось на машині із чотирнадцятьма гігабайтами оперативної пам'яті та на базі двоядерного процесору відео картою NVIDIA Tesla K80 GPU. Для тестування оберемо набір даних AG_News та спробуємо натренувати мережу з використанням кількох поширених моделей: fastText, TextCNN, TextRNN, CharCNN та Transformer. У таблиці 2.1 зведені результати замірів точності та швидкості виконання.

Таблиця 2.1

Порівняння моделей

Модель	Точність, %	Час виконання, хв.
fastText	89.46	16.0
TextCNN	88.57	17.2
TextRNN	88.07	21.5
CharCNN	87.70	13.08
Transformer	88.54	46.47

Бачимо, що fastText демонструє вищу точність за мінімального часу виконання, тому саме вона була обрана за модель для тренування

нейронної мережі. Зазначимо, що час виконання, представлений у таблиці, включає час завантаження та обробки даних, а також, запуск моделі. Також, для більшої точності порівняння можлива кастомізація параметрів тренування для досягнення критичних результатів на базі кожної з моделей.

3. ОГЛЯД РОЗРОБЛЕНОГО ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Загальна структура системи

Розроблене програмне забезпечення реалізоване у вигляді системи, що включає різноманітні клієнтські та серверні компоненти, що взаємодіють між собою за допомогою протоколу HTTP. Клієнтом є веб-сайт, що включає адміністраторський функціонал. Під серверними компонентами надалі маємо на увазі RESTful API для взаємодії із клієнтами через високорівневі абстракції (користувач, компанія, скарга тощо) та модуль збору та аналізу скарг.

Наведемо тезисний аналіз функціоналу, що виконує клієнтська частина системи:

1. Надання користувацького інтерфейсу для реєстрації та встановлення початкових даних компанії.
2. Перегляд статистики скарг, їхніх категорій.
3. Відображення списків актуальних скарг та їхніх ресурсів.
4. Наявність кабінету адміністратора, що дозволяє додавати кроки для розрішення проблем, операції редагування, видалення таких кроків.
5. Панель інструментів для запуску нових циклів збору та аналізу даних.
6. Викання асинхронних запитів до серверної частини системи, обробка відповідей та помилок.
7. Динамічне оновлення компонентів інтерфейсу.
8. Перегляд статусу оброблення даних.

Серверна частина, у свою чергу, реалізує наступний функціонал:

1. RESTful API, що надає інтерфейс доступу до сутностей системи через асинхронні запити.
2. Контролер для доступу до бази даних. Абстрактні функції для вибірки компаній, скарг та користувачів.

3. Частина API, що використовується модулями збору та обробки даних. Є закритою для публічних клієнтів.

Модуль збору даних включає наступний функціонал, що має функціонувати персистентно:

1. Аналіз роботоздатності ресурсів для збору даних.
2. Акумуляція даних про скарги із публічних API.
3. Запис у БД через комунікацію з API.

Сервіс аналізу текстових фрагментів скарг та категоризації виконує наступні функції:

1. Аналіз текст у методами NPL для виокремлення скарг з-поміж усіх відгуків.
2. Виокремлення категорій.
3. Комунікація із RESTful API для збереження поточного прогресу аналізу та запису результатів у БД.

Повна архітектура система наведена у Додатку 1 із вичерпним описом сценаріїв взаємодії між компонентами системи. Для коректного функціонування програмного комплексу необхідно підтримувати роботоздатність усіх компонентів цілодобово, проте введений протокол перевірки доступності сервісів (Health check) дозволяє обмежене користування системою за недоступності будь-якого компоненту. Така модульність забезпечує не лише надійність системи, але й можливість швидкого масштабування, розгортання та тестування від найменших складових до end-to-end перевірок. Усі сутності системи можна поділити на наступні категорії: функціональні модулі, користувачі за ролями, база даних. Зазначимо, що наведені нижче модулі є лише абстракціями та не є відокремленими сервісами із власними незалежними оточеннями. Нижче наведений розгорнутий перелік наявних функціональних модулів із коротким описанням відповідальностей:

1. Модуль акумуляції скарг. Відповідає за отримання скарг про наявні у системі компанії із публічних сервісів та відкритих джерел даних.

2. Модуль текстового аналізу скарг. Надає інтерфейс для виокремлення та категоризації зібраних скарг.
3. Модуль для комунікації із базою даних. Надає абстрактні моделі для валідації сутностей бази даних та виконання базових CRUD-операцій із наявними у ній даними.
4. Модуль аутентифікації користувачів. Ключова технологія модуля JWT passport допомагає авторизувати користувачів та надавати функціонал, що відповідає поточній ролі користувача.
5. RESTful API. Сервіс на основі NestJS, що є медіатором усіх асинхронних сигналів, що надходять за HTTP протоколом із різних компонентів системи.
6. Веб-сайт. Надає інтерфейс користувача для авторизації, створення профілю, механізмів запуску збору та оброблення даних. Ілюструє кінцеві результати аналізу.
7. Модуль збереження стану користувача на клієнті. Включає репозиторії із поточними робочими даними користувачів та контролери для високорівневих дій із ними.

Варто зазначити, що через використання мови програмування TypeScript для розробки всіх компонентів системи, є можливість перевикористання інтерфейсів сутностей, допоміжних функцій. У зв’язку із цим, потенційні розробники різних частин системи можуть взаємодіяти оперуючи спільною термінологією, що забезпечує перший рівень технологічної зрілості проекту.

На UML діаграмі із Додатку 1 проілюстровані внутрішні та зовнішні зв’язки між існуючими компонентами системи, протоколи зв’язку між ними. Зауважимо наперед, що всі складові частини системи комунікують за допомогою протоколу HTTP.

Не зважаючи на модульність системи та можливість дистанційного функціонування та масштабування окремих компонентів, у програмному пакеті є чітке розмежування доменів клієнтського та серверного коду.

Клієнтом виступає веб сайт, решта модулів – сервером. Повертаючись до UML діаграми, звертаємо увагу на компоненти, що можуть використовуватись кількома модулями. Так, доступ до RESTful API мають усі інші компоненти, оскільки він виступає посередником для обробки та делегації висорівневих запитів комунікації. Ефективність такої архітектури забезпечена чітким слідуванням методологіям і канонам будування RESTful API, а саме:

1. Спрощення через єдність інтерфейсів.
2. Можливість модифікації компонентів навіть під час функціонування системи.
3. Наочність зв’язку між компонентами та сервісними агентами.
4. Надійність в стійкості до відмов на системному рівні за виникненням відмов всередині компонентів.
5. Використання JSON формату даних для передачі між компонентами.
6. Останній пункт є важливою перевагою при використання технологій, заснованих на Javascript, оскільки JSON документ є JavaScript об’єктом, тобто основною структурою даних, що використовується в рамках системи.

Наведемо перелік функціоналу API, до якого можуть звертатися інші компоненти:

1. Реєстрування у системі нової компанії.
2. Запуск циклу збору та аналізу даних.
3. Отримання даних про результати збору та аналізу даних.
4. Пагінація великих обсягів даних.
5. Параметризація збору та аналізу даних.
6. Запис даних у БД через асинхронну вказівку.

Натомість модуль бази даних є доступним лише через RESTful API, що забезпечує достовірність даних, що там зберігається. Таким чином, у певний момент часу лише один клієнт БД має можливість вносити зміни.

Більш того, такий підхід допомагає перевикористовувати програмний код комунікації із API у різних частинах системи.

Поглиблюючись у особливості реалізації БД в рамках даної системи, можно ще раз наголосити на доцільності використання mongo як документної БЖ, до зберігає дані у звичному JSON форматі. Всередині БД можна виокремити такі сутності:

1. Компанія. Зосереджує основні дані про компанію: назва, локація, дата реєстрації, інформація про цикли аналізу тощо.
2. Елемент циклу. Включає у себе результати певного циклу збору та аналізу. Ідентифікуються по унікальному ключу компанії та даті проведення циклу.
3. Скарга. Містить текстовий вміст скарги, категорію та відповідний ідентифікатор циклу.

Дані пов’язані із першою сутністю можуть створюватись та оновлюватись лише веб-клієнтом, оскільки є результатом реєстрації та подальшої взаємодії користувача із адміністраторською панеллю. Друга сутність є суб’єктом роботи модулів збору та аналізу даних. Компонент збору акумулює відгуки у БД, звідки вони й надходять до аналітичного модуля.

Компонент аналізу даних є абстракцією системи над моделлю fastText. Модуль надає клієнту інтерфейс для параметризації механізмів категоризації.

Важливими складовими багатьох модулів є планувальники, що забезпечують синхронізацію між компонентами. Так, веб-сервіс повинен бути оповіщений про наявність оновлених даних у БД, що забезпечується локальним планувальником оновлень – сервісом, що комунікує із RESTful API кожні п’ять хвилин. Такий самий планувальник є у компоненті збору даних для отримання актуальних сигналів щодо необхідності продовження/старту нового циклу.

3.2. Алгоритми категоризації скарг

Для класифікації текстових фрагментів була обрана модель fastText. FastText – одна з найшвидших моделей класифікації тексту, запропонована дослідженнями компанії Facebook, яка дала порівнянні показники зі значно складними моделями на основі нейронної мережі. Заснована на моделі Bag of Words, оскільки саме її вихідні дані є вхідними даними для моделі fastText.

Нижче наведені деталі реалізації класифікатора за допомогою моделі fastText:

1. Використовуються попередньо навчені глобальні вектори для кодування слів, а саме, вектора GloVe.
2. Використовуються середні значення векторів слів для отримання векторів речень.
3. Використовується один прихований шар з 10 прихованими юнітами.
4. Результати прихованого шару подаються на вхід softmax класифікатору.
5. Для тренування використовується метод стохастичного градієнтного спуску.

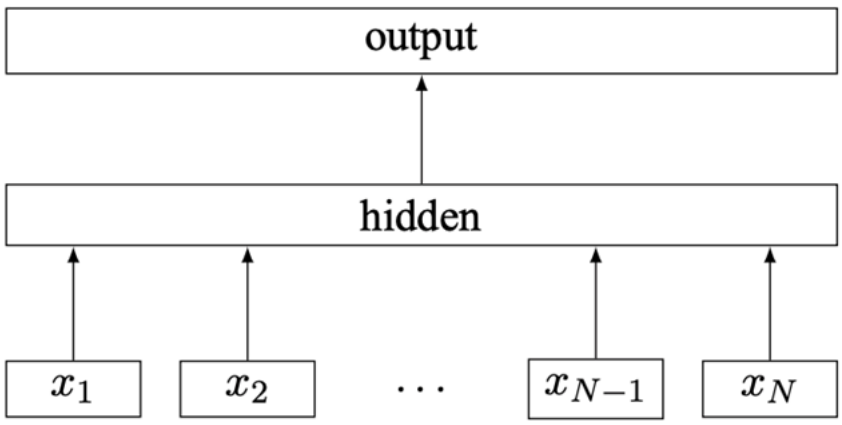


Рис. 3.1. Архітектура моделі fastText

Дана архітектура показує просту лінійну модель з рангом обмеження. Перша взвішена матриця A – це таблиця-огляд слів. Репрезентація слова потім усереднюється в текстовому поданні й подається до лінійного класифікатора.

3.3. Особливості реалізації інтерфейсу користувача

У ході створення інтерфейсу користувача основним патерном реалізації графічних елементів взаємодії стала практика Material deisgn. Для її повноцінної реалізації усі компоненти інтерфейсу були взяті із відкритої бібліотеки material-ui для React. Ці компоненти дозволяють розширити як графічний, так і функціональний арсенал звичних елементів взаємодії: кнопок, полів для введення тексту, карток, статистичних елементів, тексту, контейнерів тощо. Для оптимізації динамічного оновлення сторінок була розроблена спеціальна архітектура розміщення дочірніх компонентів в ієрархії, щоб алгоритми обходу бібліотеки mobx не оновлювали зайвих елементів. Репозиторій із даними mobx має інструменти-обгортки для швидкої передачі нових даних по графу компонентів, тому їх оптимізація стала ключовою задачею, рішення якої забезпечило плавність інтерфейсу. Більш того, плавність інтерфейсу була досягнута виокремленням усієї бізнес логіки, а саме, асинхронних запитів до API та обробки отриманих даних, у окремі сервіси, що не заморожують роботу UI-компонентів.

Важливою складовою також стала обробка помилок та виключних ситуацій, адже необхідно диференціювати клієнтські та серверні помилки й ілюструвати користувачу тип помилки. Модульність клієнтської частини допомагала виокремити місця скупчення помилок та додати в спеціальні функціональні вузли обробники. Як було сказано раніше, веб-сервіс може функціонувати за тимчасової недоступності серверу через просунуту систему збереження стану – користувач може оперувати вже наявними

даними. Велика кількість компонентів повторно використовується в системі із уже задекларованими стилізацією та функціоналом, що гнучко параметризується.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Особливості реалізації

Система реалізована у вигляді трьохрівневої архітектури, у якій клієнтом є адміністраторська панель, сервером застосунків є RESTful API та модулі збору та аналізу даних, а сервером даних є база даних MongoDB.

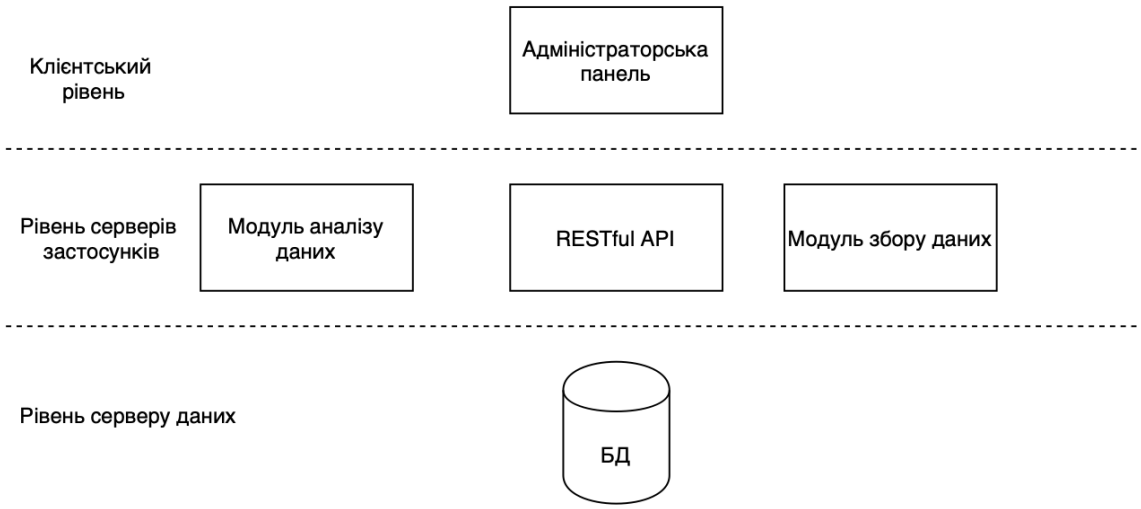


Рис. 3.1. Триярусна архітектура системи

Дана архітектура є класичним патерном розробки програмного забезпечення, що забезпечую можливість швидкого масштабування, обслуговування сервісів без зупинки усього комплексу, не вимагає педантичної уваги до швидкості каналів зв’язку між рівнями [7]. Проте така архітектура значно збільшує час розробки програмного забезпечення та зменшує швидкість розгортання. Зауважимо, що у зв’язку із бізнес вимогами до системи, додатково витрачений час є раціональним у контексті крайньої важливості персистентності системи.

Потенціал для масштабовності системи досягнуто у тому числі й завдяки реалізації шаблонів проектування: “Стратегія”, “Абстрактна фабрика”, “Декоратор” тощо. Патерн “Абстрактна фабрика” був використаний на клієнтській частині для створення й

ін’єкцій (dependency injection) сутностей контролерів та репозиторіїв, логіка яких інкапсульвона бібліотекою inversify [8]. “Стратегія” використовувалась для визначення компонентів для рендерингу під час виконання коду на основі зовнішніх факторів. “Декоратор” є стандартним рекомендованим шаблоном у оточенні React, що використовується для доповнення компонентів необхідними властивостями для реалізації бізнес логіки та відображення графічним елементів.

Експерементальним рішенням у ході розроблення клієнтської частини стало використання dependency injection для підтримки стану компонентів. Відмовлення від стандартної бібліотеки Redux, що стала квінтесенцією функціонального програмування у React, та перехід до mobx для state management, дозволило утилізувати переваги SOLID принципів для подальшого масштабування та гармонійного розділення відповідальності між функціональними вузлами системи. Так, вдалося побудувати чітку ієрархію залежностей, кожна з яких наслідується від наперед визначених інтерфейсів, у якій кожному компоненту відповідає певний контроллер, що, у свою чергу, містить репозиторій із даними. Даний підхід реалізує SOLID-принципи розмежування інтерфейсів, інверсії залежностей тощо. Наприклад, компонент Auth має доступ до контролеру Company, що є єдиним медіатором для запису даних у відповідний репозиторій. При такому підході не порушено жодного принципу ООП, адже репозиторій є інкапсульованим у контроллері та наслідується від визначего інтерфейса. Кожен компонент у React наслідується від базового компонента й кожна залежність у системі реалізує інтерфейс.

4.2. Дизайн та вміст веб-сторінок

Базовим інструментом реалізації інтерфейсу користувача стала бібліотека material-ui, що надає комплекс компонентів для React, що реалізують однойменну дизайн-парадигму від Google. Нижче наведені

основні компоненти бібліотеки, що були використані для побудови графічних елементів:

1. Typography. Компонент для відмальовування текстових фрагментів.
2. TextField. Компонент для створення полей вводу даних.
3. Card. Контейнер із тінями для відображення блоків даних. Є базовим компонентом-обгорткою, що містить решту графічних елементів.
4. CardHeader. Скдадова частина компоненту Card. Є заголовком блоку компонентів.
5. CardActionArea. Зона компоненту Card із можливістю інтеракції із користувачем.
6. Button. Дизайнерська реалізації кнопки від material-ui. У системі використовується outline версія компоненту.
7. ExpansionPanel. Елемент із можливістю розгортання деталей. Використовується для додавання коментарю до циклу збору і аналізу даних під час його створення.
8. LinearProgress. Елемент прогресу, що використовується для індикації процесу виконання певного етапу. Виглядає як бігуча стрічка певного кольору.
9. ColorCircularProgress. Виконує ту саму функцію, що й попередній елемент, але із круговим дизайном.
10. Grid. Компонент розмітки елементів на сторінці.

Усі сторінки адміністраторської панелі розділені на відкриті та приватні. Відкритою є сторінка із формою авторизації для входу/реєстрації нової компанії у системі (рис. 4.1). Дана сторінка динамічно оновлюється залежно від сценарію користувача (авторизація або реєстрація) та підключена до єдиного сервісу Auth, що відповідає за дії авторизації, інкапсульовані від інших сутностей системи. Після реєстрації нової компанії користувач має пройти обов’язковий етап заповнення початкових

даних, що є кореневими у подальшому функціонуванні системи. Ці дані можливо заповнити на сторінці, проілюстрований на рисунку 4.2. Після проходження початкового налаштування або після авторизації при іншому сценарії взаємодії, користувач перенаправляється на головну дошку застосунку (рис. 4.3). Вона складається із навігаційної панелі у верхній частині сторінки, за допомогою якої відбувається перемикання між трьома основними компонентами: панеллю контролю, списком циклів та налаштуваннями компанії. Перемикання виконується із кастомізованою анімацією.

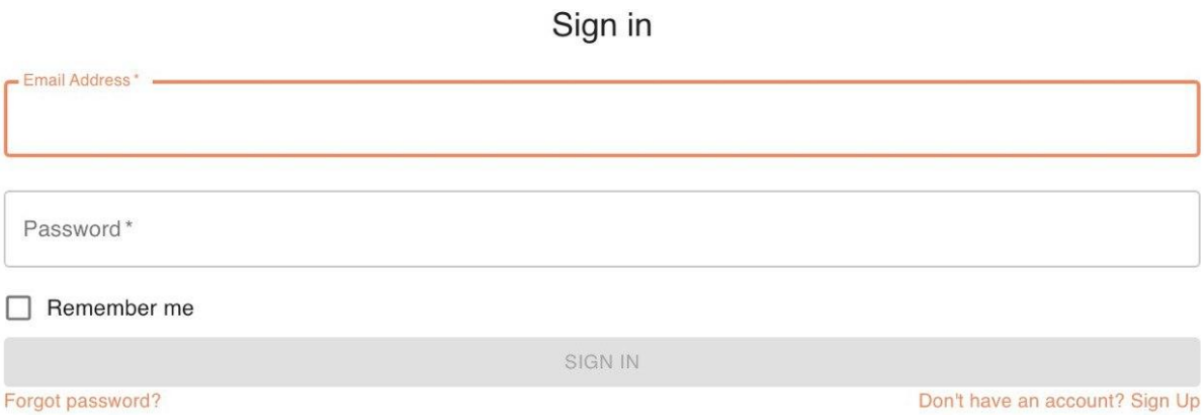


Рис. 4.1. Форма авторизації

Компонент панелі контролю містить діаграму, що ілюструє відсоткові частини категорій скарг після останнього циклу збору й аналізу даних. Також, панель містить список останніх виявлених скарг та список категорій скарг. У верхній частині компоненту знаходиться панель для контролю виконання етапів життєвого циклу системи. Виконання певного етапу супроводжується відповідною анімацією, може бути призупинений користувачем, а також ілюструє статус завершення виконання. При натисканні на картку із останніми скаргами можно побачити розгорнутий коментар із додатковими даними у вигляді модального вікна. У вкладці Cycles можна переглянути історію виконаних циклів збору та аналізу скарг та переглянути їх деталі та статус. Також є можливість ініціалізації нового

циклу після натискання на кнопку “Add Cycle”. Після цього сторінка динамічно перевантажиться і з’явиться форма заповнення даних про цикл. При розгортанні панелі категорії скарги є можливість додати коментар із виконаними діями, що очікувано вирішать проблему у наступному циклі.

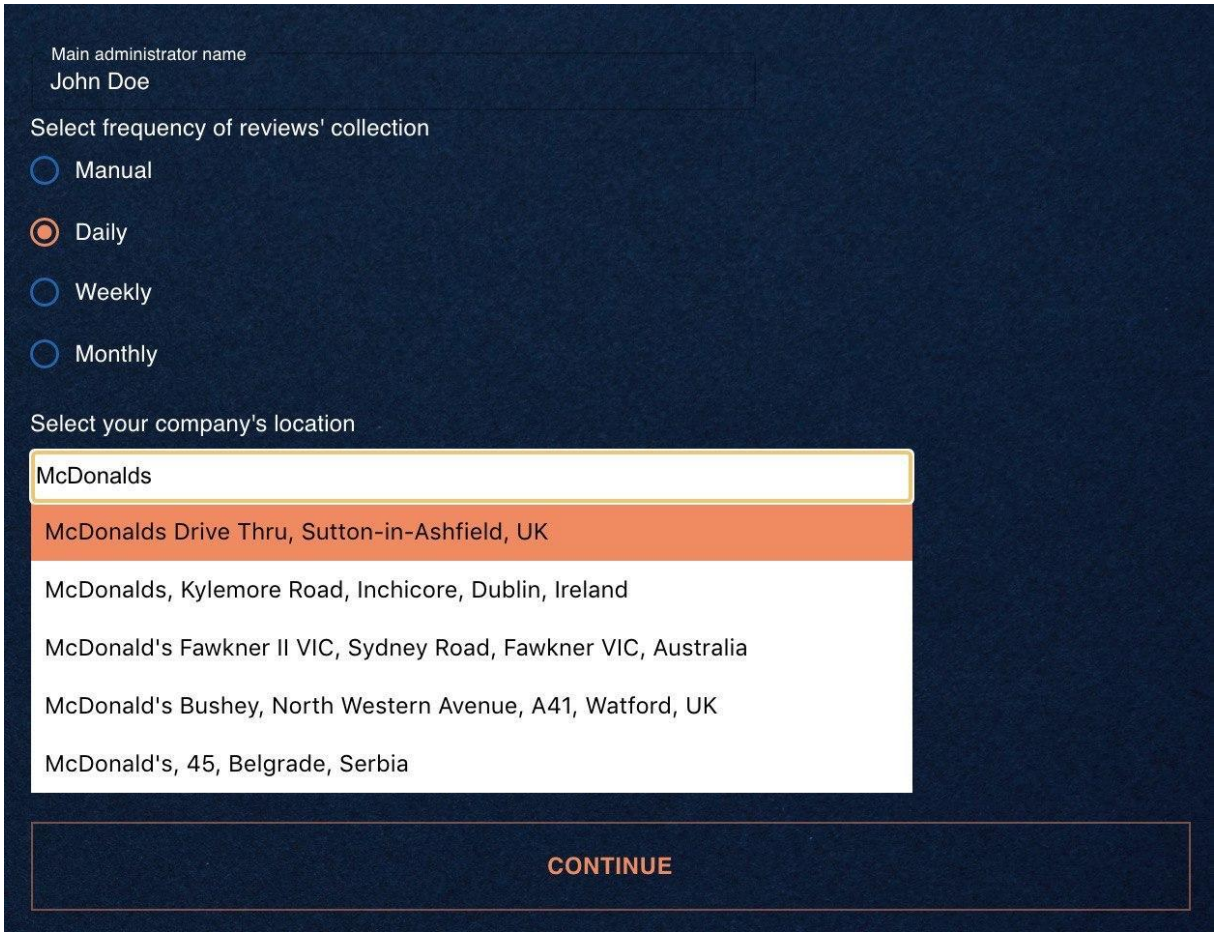


Рис. 4.2. Форма заповнення початкових даних

Вкладка налаштування дозволяє головному адміністратору змінити базові дані про компанію та змінити періодичність виконання циклів аналізу.

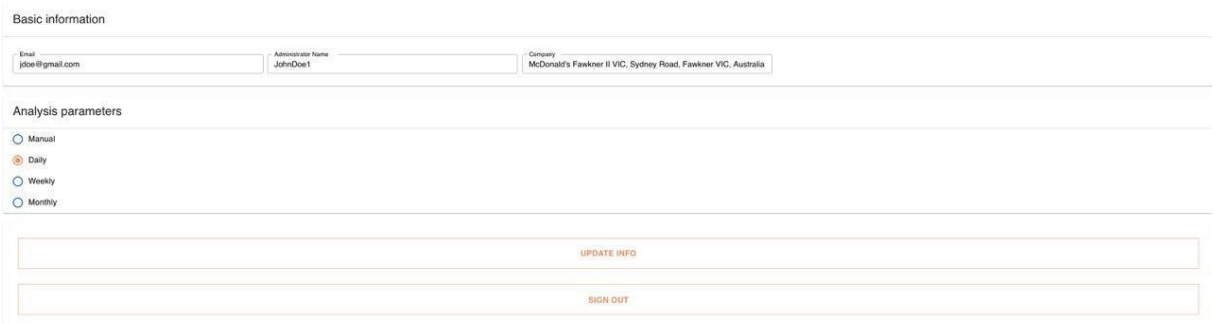


Рис. 4.3. Сторінка налаштувань

Головним компонентом, що відповідає за відображення статусу виконання циклу виконання збору та аналізу є Pipeline (рис. 4.4). Кожному кроку виконання відповідає колір та текстове уточнення. Після виконання циклу користувач может переглянути результат виконання кожного окремого етапу та ідентифікувати проблемний етап у разі технічного збою.

Рис. 4.4. Компонент Pipeline

Наступним елементом у компонентному дереві головної сторінки є колова діаграма категорій виявлених проблем (рис. 4.5). Кожній категорії відповідає колір на діаграмі, а за відсутності актуальних даних, замість діаграми відображатиметься компонент-замінник.

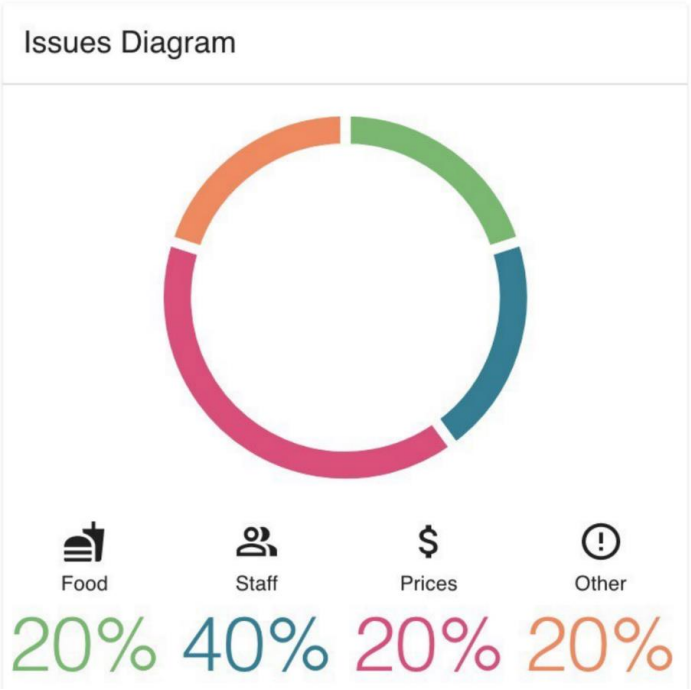


Рис. 4.5. Компонент Pipeline

Поруч із круговою діаграмою розташована лінійна діаграма (рис. 4.6), натискаючи на яку користувач може переглянути модальне вікно із усіма скаргами із певної категорії.

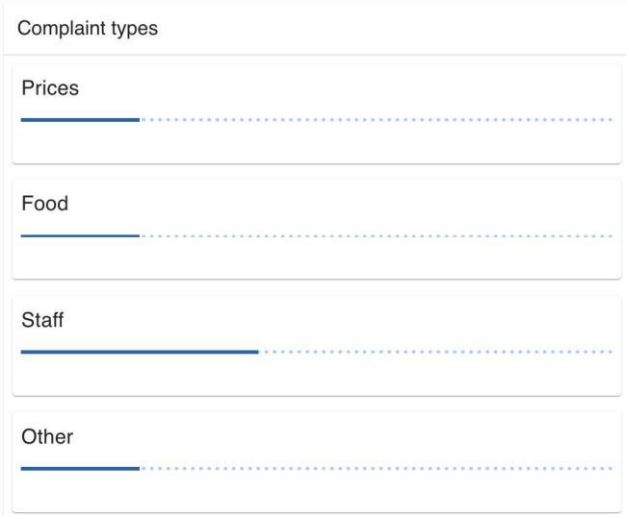


Рис. 4.6. Лінійна діаграма

4.3. Реалізація RESTful API

Серверна частина вичерпно реалізує основні приципи побудови RESTful API:

1. Уніфікований інтерфейс.
2. Відсутність стану.
3. Кешовність.
4. Багаторівнева система.
5. Код на вимогу

Ці принципи, разом із функціоналом, якого потребували функціональні вимоги, були досягнуті за допомогою влаштованого інструментарію фреймворку NestJS [10]. Наступні компоненти бібліотеки були імплементовані при реалізації RESTful API:

1. Контролери. Використовувались для розмежування сутностей на високому рівні абстракції. У системі було реалізовано Company, Complaint, Schedule контролери для доступу за типізованими шляхами api/v1/company/…, api/v1/complaint/…, api/v1/schedule тощо.
2. Провайдери. Використовувались для ін’єкції залежностей у контролери на глобальному рівні.
3. Модулі. Глобальні компоненти, що декларують залежності при ініціалізації сервера.
4. Pipes (Validation-Pipe). Використовувались для валідації вхідних запитів. Наприклад, payload для реєстрації нової компанії має містити коректні дані. У разі відсутності або хибності даних, сервер повертає помилку 400.
5. Гарди. Використовувались для валідації ролі користувача під час запиту. Так, є ендпоінти, доступні лише для певного типу клієнтів. При спробі доступу до такого ресурсу забороненим клієнтом або без JWT-токену, клієнт отримає відповідь із помилкою 401.

Усі функціональні шляхи були розділені на три високорівневі контролери:

1. Companies. Містить функції для CRUD-операцій із сутностями типу Company.
2. Cycles. Відповідають за створення та запуск циклів збору та аналізу даних.
3. Complaints. Містить функції для CRUD-операцій із сутностями типу Complaint.

Нижче наведений повний перелік реалізованих точок вход для API:

1. GET company/:id.
2. POST company/register.
3. POST company/login.
4. PUT company/complete-onboarding.
5. PUT company/update.
6. POST company/delete.
7. POST cycles/add.
8. POST cycles/start/:id.
9. GET cycles/for-company/:id.
10. POST complaints/add.
11. GET complaints/for-cycle.

Із точки зору API робота із системою виглядає наступним чином. Адміністратор компанії посилає початкові дані на company/register та отримує відповідь із статусом 201 у разі успішної реєстрації.

Далі, користувач має авторизуватися, надіславши зареєстровану адресу та коректний пароль на company/login та отримати токен авторизації. Даний токен надає користувачу доступ до інших точок входу API. Обов’язковим кроком є надсилання початкових даних про компанію на company/complete-onboarding, проте, даний крок регулюється клієнтом, а сервер не знає про обмеження. Далі, користувач має доступ до всіх точок входу контролерів companies, cycles та complaints.

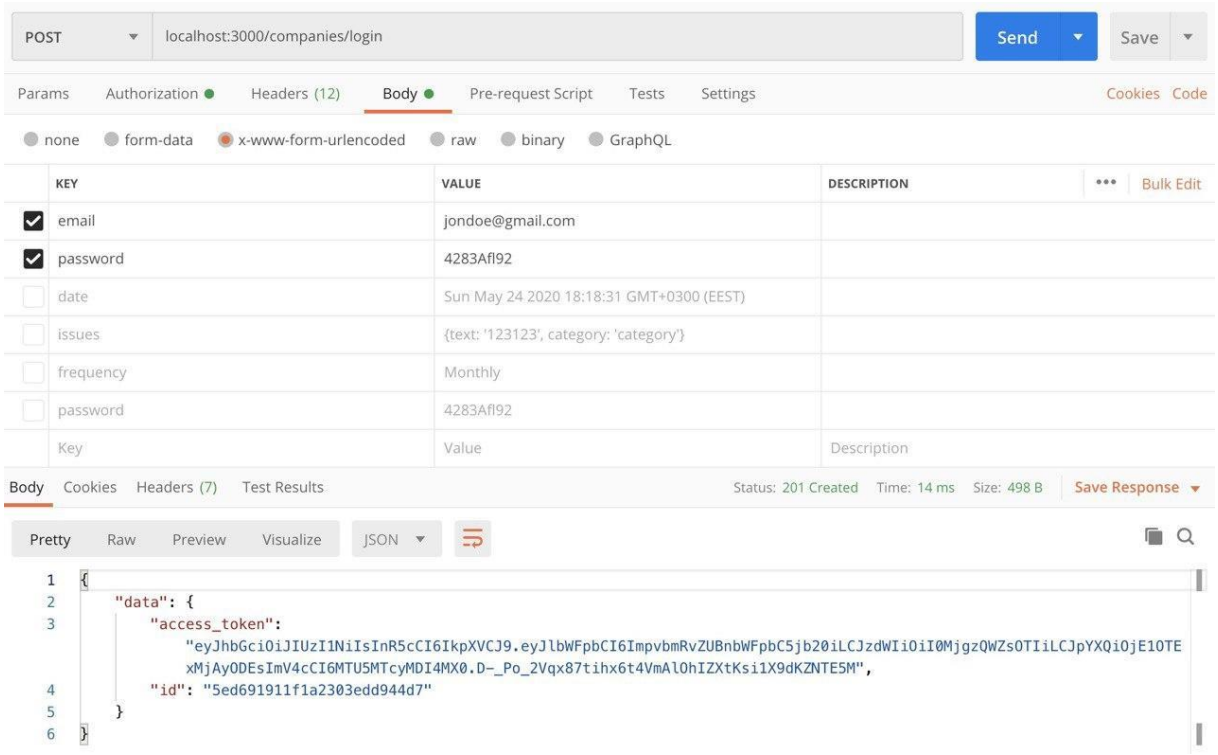
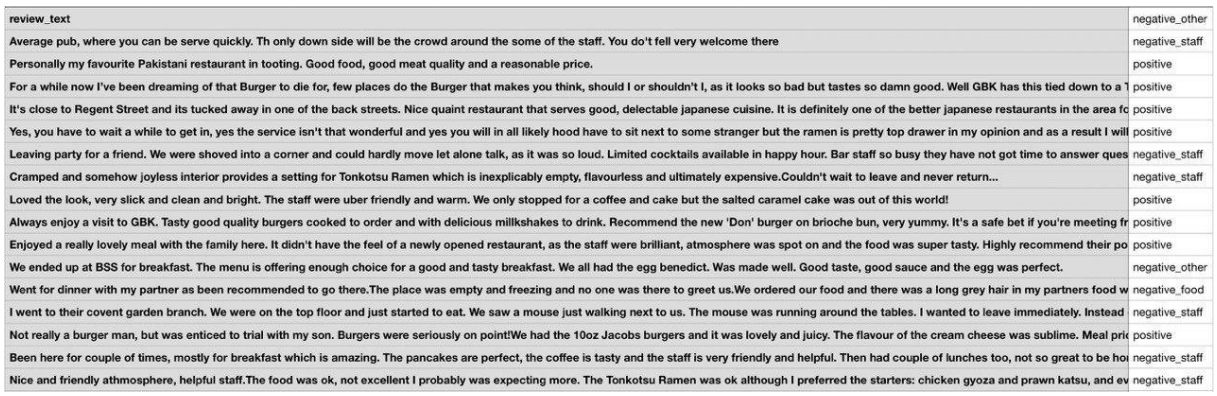


Рис. 4.7. Приклад запиту та відповіді для авторизації

4.4. Реалізація модуля класифікації скарг

Після вибору моделі fastText постала проблема вибору класів для подальшого створення класифікатору. Для тренування стало необхідним розмічення вибіркового набору відгуків мануально, для того, щоб створити референс для майбутньої мережі. Для цього був обраний дата сет із десяти тисяч відгуків і підготований для подальшого маркування. Для цього були виокремлені дві колонки: із текстом відгуку та пустою колонкою для категорії. Усі відгуки із рейтингом більше трьох були автоматизовано промарковані Positive класом.



review_text	
Average pub, where you can be serve quickly. Th only down side will be the crowd around the some of the staff. You do't fell very welcome there	negative_staff
Personally my favourite Pakistani restaurant in tooting. Good food, good meat quality and a reasonable price.	positive
For a while now I've been dreaming of that Burger to die for, few places do the Burger that makes you think, should I or shouldn't I, as it looks so bad but tastes so damn good. Well GBK has this tied down to a T	positive
It's close to Regent Street and its tucked away in one of the back streets. Nice quaint restaurant that serves good, delectable japanese cuisine. It is definitely one of the better japanese restaurants in the area fo	positive
Yes, you have to wait a while to get in, yes the service isn't that wonderful and yes you will in all likely hood have to sit next to some stranger but the ramen is pretty top drawer in my opinion and as a result I will	positive
Leaving party for a friend. We were shoved into a corner and could hardly move let alone talk, as it was so loud. Limited cocktails available in happy hour. Bar staff so busy they have not got time to answer ques	negative_staff
Cramped and somehow joyless interior provides a setting for Tonkotsu Ramen which is inexplicably empty, flavourless and ultimately expensive.Couldn't wait to leave and never return...	negative_staff
Loved the look, very slick and clean and bright. The staff were uber friendly and warm. We only stopped for a coffee and cake but the salted caramel cake was out of this world!	positive
Always enjoy a visit to GBK. Tasty good quality burgers cooked to order and with delicious millkshakes to drink. Recommend the new 'Don' burger on brioche bun, very yummy. It's a safe bet if you're meeting fr	positive
Enjoyed a really lovely meal with the family here. It didn't have the feel of a newly opened restaurant, as the staff were brilliant, atmosphere was spot on and the food was super tasty. Highly recommend their po	positive
We ended up at BSS for breakfast. The menu is offering enough choice for a good and tasty breakfast. We all had the egg benedict. Was made well. Good taste, good sauce and the egg was perfect.	negative_other
Went for dinner with my partner as been recommended to go there.The place was empty and freezing and no one was there to greet us.We ordered our food and there was a long grey hair in my partners food w	negative_food
I went to their covent garden branch. We were on the top floor and just started to eat. We saw a mouse just walking next to us. The mouse was running around the tables. I wanted to leave immediately. Instead	negative_staff
Not really a burger man, but was enticed to trial with my son. Burgers were seriously on point!We had the 10oz Jacobs burgers and it was lovely and juicy. The flavour of the cream cheese was sublime. Meal pric	positive
Been here for couple of times, mostly for breakfast which is amazing. The pancakes are perfect, the coffee is tasty and the staff is very friendly and helpful. Then had couple of lunches too, not so great to be hor	negative_staff
Nice and friendly athmosphere, helpful staff.The food was ok, not excellent I probably was expecting more. The Tonkotsu Ramen was ok although I preferred the starters: chicken gyoza and prawn katsu, and ev	negative_staff

Рис. 4.8. Приклад мануального маркування відгуків

Для підтвердження концепції були обрані наступні початкові класи:

1. Positive – позитивні відгуки, що не оброблюються системою.
2. Negative_price – негативні відгуки, пов’язані із ціноутвореннями у закладах.
3. Negative_food – скарги пов’язані із якістю їжі у закладах.
4. Negative_staff – негативні відгуки, пов’язані із сервісом та персоналом.
5. Negative_other – решта негативних відгуків, що не потрапили в попередні категорії.

Усі мануально промарковані відгуки були зведені в окремий файл формату csv. Приклад такого маркування можна побачити на рисунку 4.6.

Варто зазначити, що виявлення класу *Positive* відбувалося без участі машинного навчання, і було виконано керуючись допущенням, що усі відгуки із користувацькою оцінкою меншою за 3/5 вважається скаргою.

Наступним кроком є подання промаркованого набору скарг на вхід моделі fastText разом із наперед тренованими GloVe векторами слів [11]. Після кількох епох тренувань такої мережі, вихідним результатом є шуканий класифікатор. Після налаштування моделі було проведення тестування основних показників: точності, повноти, F-міри та підтримки. У таблиці 4.1 зведені результати такого тестування.

Таблиця 4.1

Тестування нейронної мережі

	Precision	Recall	F1-score	Support
food	0.72	0.81	0.77	70
other	0.81	0.86	0.83	50
price	0.94	0.66	0.77	44
staff	0.89	0.91	0.90	75

Фінальна точності класифікатора становить 82% на заданих тестових даних. Після тестування була побудована матриця хибності (рис. 4.8), що вказує на точність визначення категорії. У ідеальних умовах значення мають бути розташовані винятково на діагоналі, а решта елементів матриці дорівнюють нулю.

Проте у реальних умовах може існувати певна кількість фрагментів, що не були класифіковані коректно. Це можно побачити по значенням у недіагональних елементів матриці. Ці значення вказують на кількість коментарів із іншого класу, що були віднесенні в інший клас через недостатнє тренування моделі.

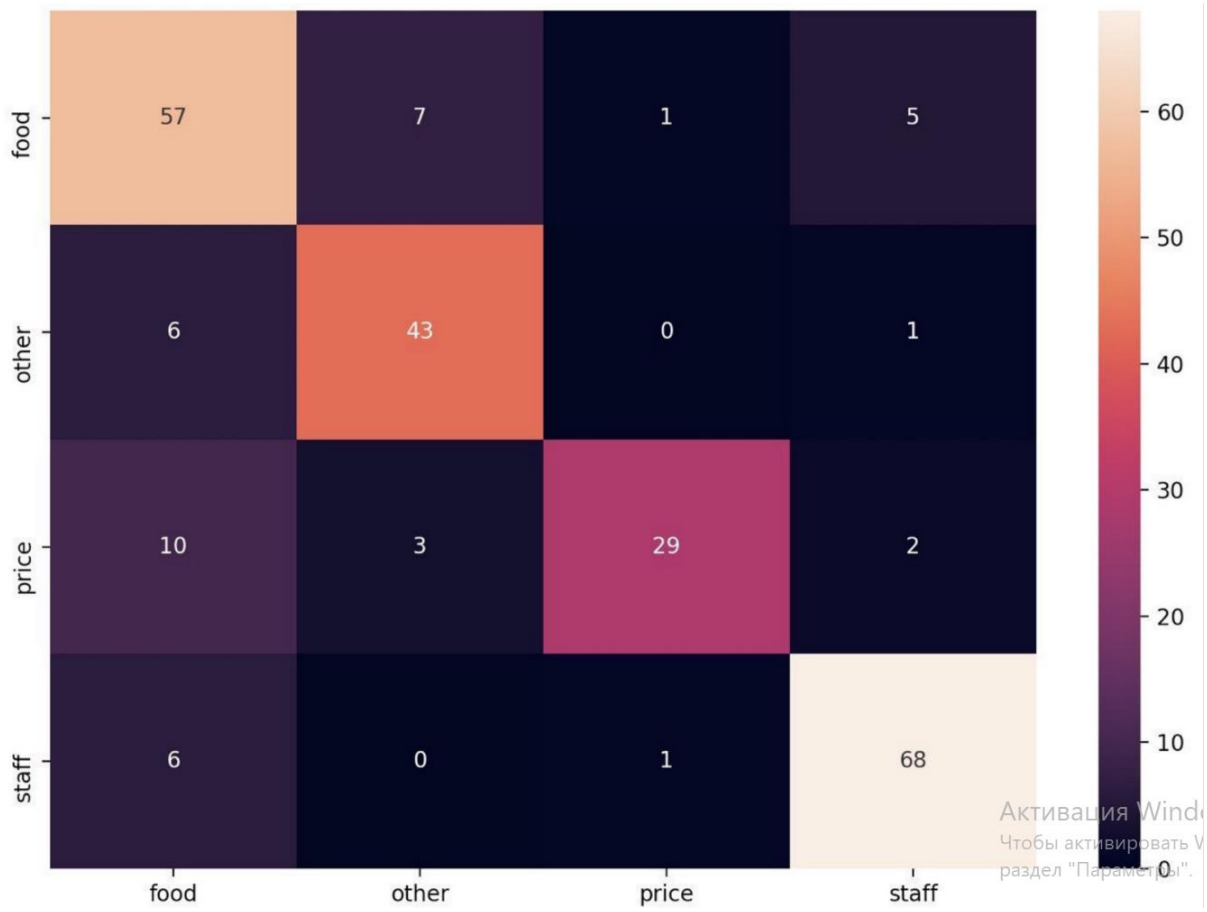


Рис. 4.8. Матриця хибності

4.5. Тестування системи

Тестування даної системи передбачає повний комплекс заходів по Unit-тестуванню, інтеграційним перевіркам та end-to-end переглядів для автоматизованого виявлення локальних та глобальних проблем компонентів програмної системи. Для Unit-тестування клієнтської частини рекомендується фреймворк Jest із інструментом Enzyme. Дані бібліотеки дозволять тестувати граничні випадки використання системи та перевіряти коректність рендерингу компонентів на етапі їх приєднання до DOM-дерева. Інтеграційні тести рекомендовано виконувати із застосування інструменту CucumberJS, що єдиний у своєму класі має підтримку Typescript. Даний інструмент дозволить протестувати серверну частину застосунку абстрагуючись від сценаріїв клієнтського досвіду. Таке тестування виявляє потенційні проблему серверу на етапі масштабування. Розповсюдженим для end-to-end тестування веб-додатків є фреймвор

Cypress, що дозволяє імітувати взаємодію користувача із адміністраторською панеллю у режимі реального часу із ілюстрацією кожного кроку виконання. Результати усіх перевірок рекомендується зберігати у вигляді Allure звіту для більш наглядного представлення для розробників.

У таблиці нижче зведемо базові тестові випадки, що можуть стати базисом як для мануального тестування, так і покроковим сценарієм для end-to-end або інтеграційних тестів високого рівня:

Таблиця 4.2

Набір тестових випадків

Тестовий крок	Очікуваний результат
Користувач заповнює форму авторизації некоректними даними	Кнопка “Sign In” є неактивною, поля із некоректними значеннями виділені
Користувач заповнює форму реєстрації некоректними даними.	Кнопка “Sign Up” є неактивною, поля із некоректними значеннями виділені
Користувач заповнює форму реєстрації коректними даними та натискає на кнопку “Sign Up”	Форма перемикається у режим авторизації, користувач бачить модальне вікно із повідомленням про успішну реєстрацію
Користувач заповнює форму авторизації коректними даними та натискає на кнопку “Sign In”	Користувач напрявляється на сторінку заповнення початкових даних
Користувач заповнює форму заповнення коректними даними та натискає на кнопку “Continue”	Користувач направляється на основну сторінку із статистикою. Статистичні елементи порожні у випадку первої авторизації
Користувач перемикається на вкладку “Cycles”	Контент сторінки динамічно оновлюється з анімацією. Користувач бачить порожній список циклів

Продовження табл. 4.2

Користувач натискає на кнопку “Add cycle”	Зміст веб-сторінки динамічно оновлюється й з’являється форма створення циклу
Користувач заповнює форму із параметрами циклу коректними даними та натискає на кнопку “Add”	Сторінка знов оновлюється й користувач бачить створенний цикл у списку циклів
Користувач натискає на кнопку запуску циклу	Кнопка запуску циклу стає неактивною
Користувач перемикається на вкладку "Control Room" при активному циклі	Компонент “Pipeline” вказує на поточний етап виконання циклу збору та аналізу скарг. Після успішного виконання циклу статус компонента стає *completed.* Компоненти відображення результатів наповнені статистичними даними
Користувач перемикається на вкладку “Settings”	Контент основної сторінки оновлюється та з’являється форма із даними про компанію та налаштуваннями циклу
Користувач змінює дані у полях та натискає кнопку “Update information”	З’являється модальне вікно-підтвердження зміни даних компаніх. Дані компанії у системі збережені
Користувач натискає на кнопку “Sign Out”	Сесія користувача завершується. Виконується пере направлення користувача на сторінку авторизації/реєстрації
Користувач авторизується в системі, попередньо заповнивши початкові дані про компанії при першій авторизації	Користувач не перенаправляється на сторінку заповнення початкових даних про компанію. Навігація відбувається на головну сторінку із вімкненою вкладкою “Control Room”

4.6. Рекомендації щодо подальшого використання системи

Користувачеві необхідно розуміти, що дана система є ефективною лишу у комплексі з раціональними адміністраторськими діями персоналу. Так, розроблене ПО повинне слугувати інструментом, що лише координує дії та надає доступ до відфільтрованих даних та історії дій. Варто зауважити, що збій у роботі будь-якого компоненту системи має бути зафіксований обслуговуючим персоналом та повідомлений розробникам системи для корекції. Адміністраторський веб-сайт підтримується усіма сучасними браузерами, тому рекомендовано не використовувати старі версії розповсюджених браузерів для запобігання непередбачених проблем із користувацьким інтерфейсом. Інших вимог до апаратної частини машини немає.

4.7. Можливості до масштабування та вдосконалення системи

Оскільки розроблена система може використовуватись у різних сферах бізнесу, для категоризації скарг необхідно розробити більшу кількість наперед визначених класів. Так, у даному ПЗ були обрані класи для ідентифікування проблем із цінами, їжою, персоналом та усі інші. Такі категорії були обрані, щоб забезпечити коректну роботу системи для власників ресторані та готелів, що є найпопулярнішими клієнтами. Дійсно, для того, щоб натренувати модель розрізняти певний клас скарг, необхідно провести затратні у часі заходи по мануальному маркування скарг цим класом. Іншими словами, необхідно в ручну вибрати близько тисячі коментарів певного класу, щоб певна категорія стала функціонуючою у системі. Для того, щоб поповнювати промарковану колекцію системи, пропонується використовувати широко поширений метод, що полягає у тому, що кожен користувач обирає клас випадково обраного коментаря для того, щоб авторизуватися у системи. Базуючись на такій техніці, Google збирає дата сет промаркованих зображень через Captcha.

Також, вагомим покращенням стане розширення джерел для агрегації відгуків. На даний момент система збирає дані із обгортки над Google Places API, що називається Wextractor. З іншого боку, існує багато інших сервісів-агрегаторів: Yelp, Booking тощо. Проте, на момент створення даного ПЗ, вони не надають функціонально вичерпного API для збору необмеженої кількості коментарів. Проте у майбутньому вони можуть бути використані для суттєвого покращення програмного комплексу.

Слід зауважити, що для будь-якої сучасної веб-інфраструктури важливою є гнучкість та повнота параметризації виконуваних процесів. Тому розробленній системі бракує динамічних

механізмів для модифікування циклу під час його виконання. Таким чином, можливим стане зупинка, продовження або перезапуск циклу. На даний момент, такі можливості відсутні. Також, вагомою перевагою може стати розширена параметризація циклів виявлення та оброблення скарг. Адміністратору може знадобитися можливість прикріплення певних скарг до циклу для створення референту для аналізу результатів. Більш того, раціональним буде надання користувачу вибору джерел, з яких агрегуються відгуки.

Кажучи про масштабованість, наступним кроком є додання декількох адміністраторів для моніторингу скарг однієї компанії. Технічно реалізованою така можливість може стати через модифікацію структури бази даних та створення сутності User із залежністю one-to-may відносно компанії. Також, суттєвого покращення до моніторингової системи може принести розширення можливих клієнтських додатків системи. Наприклад, можлива імплементація мобільного додатка із аналогічним до адміністраторського веб-додатка функціоналом. Зауважимо, що рекомендованим фреймворком для створення мобільного застосунку є React Native, що слідує таким самим парадигмам і шаблонам, як і бібліотека React на боці веб-сервісу. Таким чином, більшість клієнтських

графічних та функціональних компонентів, сервісів та інтерфейсів можуть бути перенесені в мобільний застосунок із мінімальними змінами коду. Даний фреймворк надає можливості для кросплатформеної розробки мобільних застосунків, що можуть виконуватись на мобільних операційних системах Android та IOS.

ВИСНОВКИ

Метою даного дипломного проєкту була реалізація веб-сервісу для автоматизованого виявлення та оброблення скарг Інтернет-користувачів.

Перед початком розроблення даного програмного комплексу був проведений збір функціональних та не функціональних вимов до ПЗ, на основі якого був сформований список засобів розробки, термінів проектування та вигляд кінцевої документації. Набір технологій, алгоритмів та сторонніх бібліотек виявився найоптимальнішим для створення сучасної веб-системи із клієнтськими та серверними за стосунками, що були успішно інтегровані із алгоритмічною складовою розробленого класифікатору.

Представлений програмне рішення включає:

1. Клієнтський додаток у вигляді адміністраторської панелі, що надає високорівневий доступ до функціоналу системи для користувачів.
2. RESTful API для встановлення комунікації між компонентами системи та БД.
3. Модуль збору даних із публічних API.
4. Класифікатор для встановлення категорій скарг.

У результаті розроблення були виконані всі функціональні та нефункціональні вимоги, висунуті на етапі планування, тестування проведено у повному обсязі відповідно до затвердженої методики тестування.

Інтеграція вихідної системи у сучасний бізнес, що функціонує у сфера послуг, вирішує актуальну проблему автоматизації та, відповідно, оптимізації операційних процесів, що стосуються збору, аналізу та реагування на скарги користувачів.

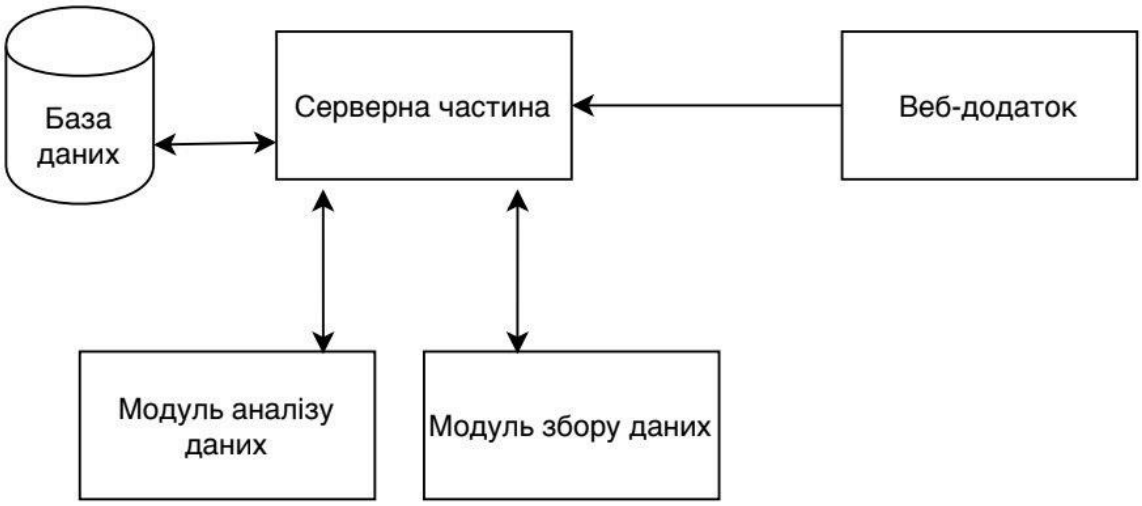
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. JavaScipt Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/uk/docs/Web/JavaScript. Дата доступу: травень 2020. Назва з екрану.
2. TypeScript Documentation [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/handbook. Дата доступу: травень 2020. Назва з екрану.
3. ExpressJS Documentation [Електронний ресурс]. – Режим доступу: http://expressjs.com/. Дата доступу: травень 2020. Назва з екрану.
4. NestJS Documentation [Електронний ресурс]. – Режим доступу: https://docs.nestjs.com/. Дата доступу: травень 2020. Назва з екрану.
5. AngularJS Documentation [Електронний ресурс]. – Режим доступу: https://angularjs.org/. Дата доступу: травень 2020. Назва з екрану.
6. VueJS Introduction [Електронний ресурс]. – Режим доступу: https://vuejs.org/v2/guide/. Дата доступу: травень 2020. Назва з екрану.
7. React Getting Started [Електронний ресурс]. – Режим доступу: https://reactjs.org/docs/getting-started.html. Дата доступу: травень 2020. Назва з екрану.
8. Three-Tier Architecture [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/24649/three-tier-architecture. Дата доступу: травень 2020. Назва з екрану.
9. Inversify Github. The Container API [Електронний ресурс]. – Режим доступу: https://github.com/inversify/InversifyJS/blob/master/wiki/container_api.md. Дата доступу: травень 2020. Назва з екрану.
10. RESTful API Design – Step By Step Guide [Електронний ресурс]. – Режим доступу: https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fcdbf. Дата доступу: травень 2020. Назва з екрану.
11. Armand Joulinm. Bag of Tricks for Efficient Text Classification / Joulinm Armand, Grave Edouard, Bojanowski Piotr, Mikolov Tomas. – 2016. – 3 p.

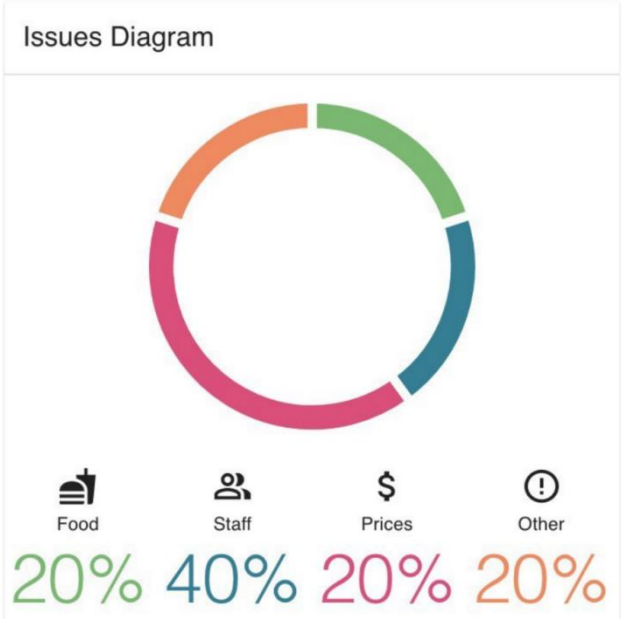
ДОДАТКИ

Додаток 1

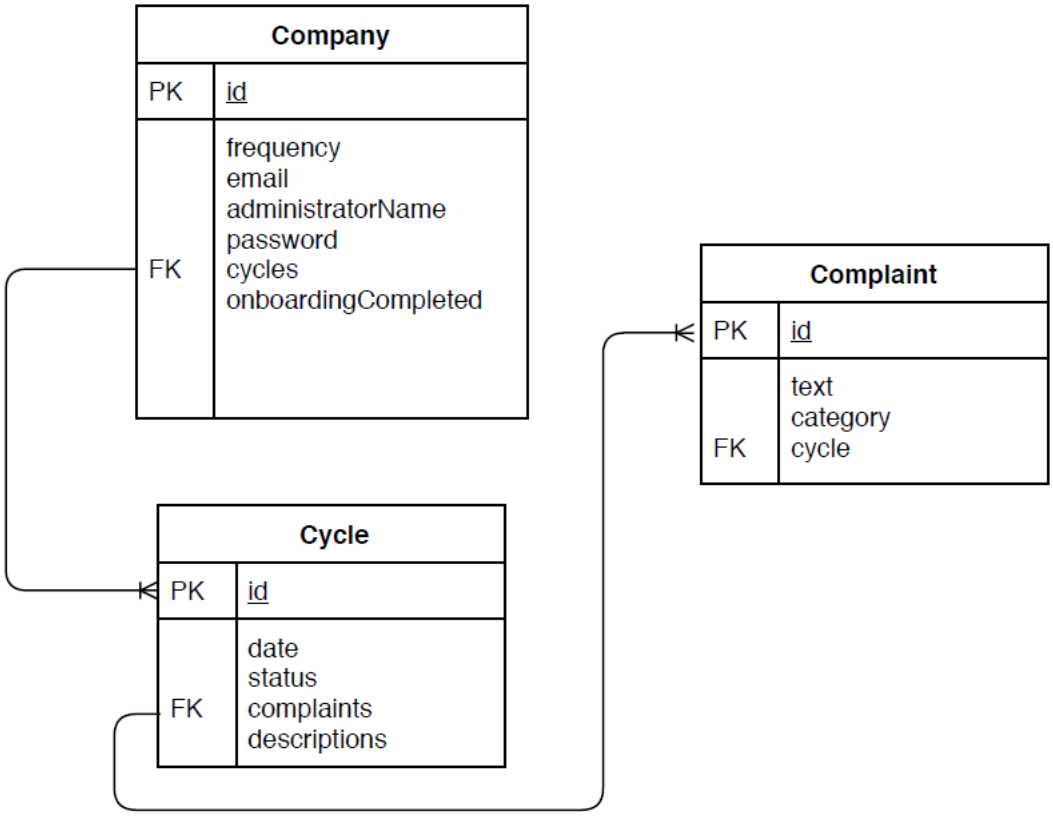
Копії графічних матеріалів



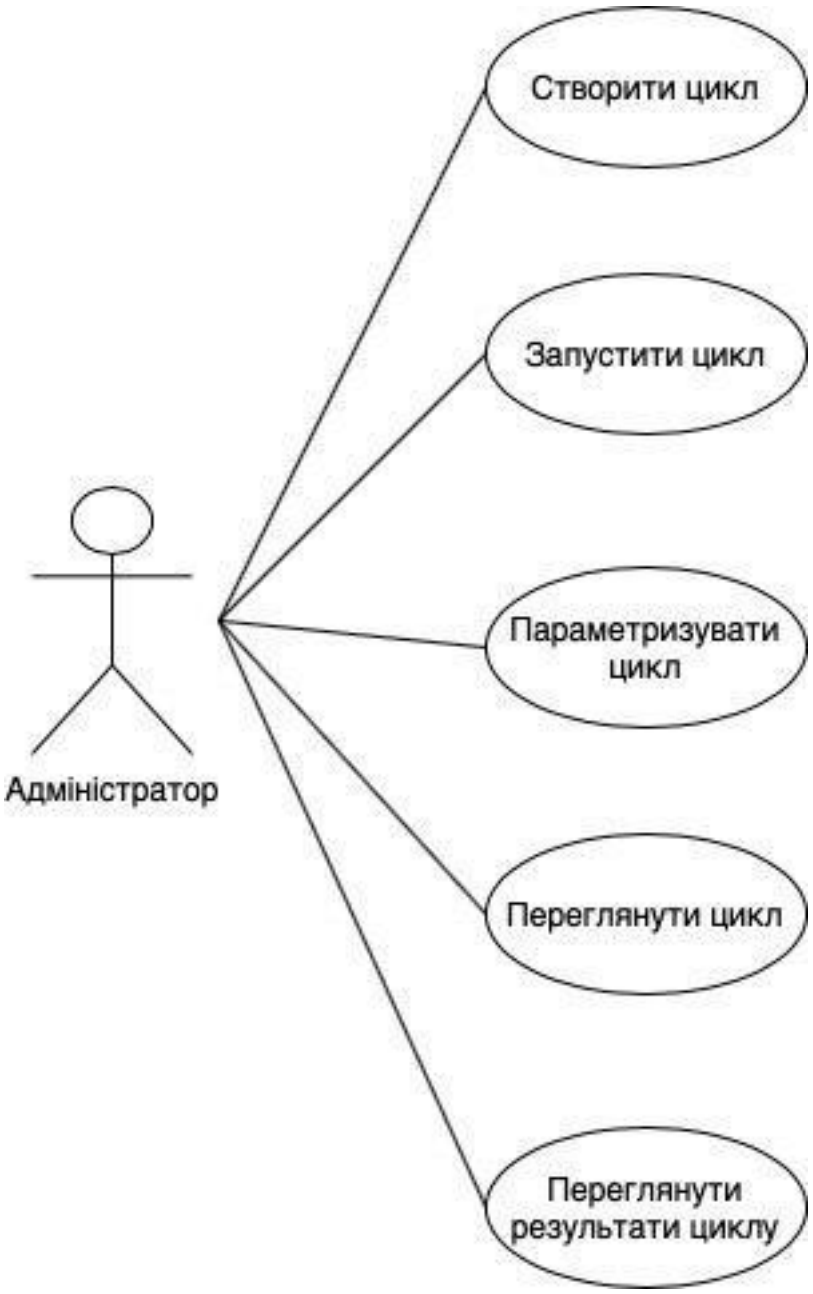
Герасимов Артем Сергійович
гр. КП-61



Герасимов Артем Сергійович
гр. КП-61



ДП.045440-06-99
Веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів. Структура бази даних. ERD-діаграма



ДП.045440-07-99
Веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів. Функціональність програмних засобів. Діаграма прецедентів

Додаток 2

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ТА ОБРОБЛЕННЯ СКАРГ ІНТЕРНЕТ-КОРИСТУВАЧІВ

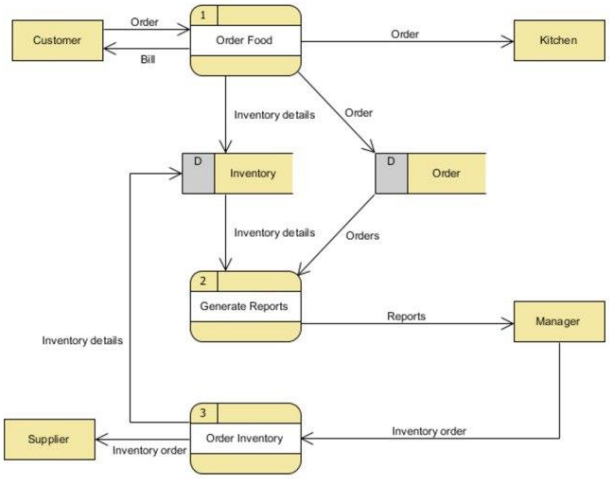
Виконав: Герасимов Артем Сергійович
Керівник: Доцент кафедри ПЗКС, к.т.н., Заболотня Тетяна Миколаївна

Київ – 2020

АКТУАЛЬНІСТЬ

Лише за 2018 рік кількість закладів швидкого харчування в Україні збільшилась на 2,7 тис. одиниць.

АКТУАЛЬНІСТЬ



Бачимо відсутність вузла обробки скарг у типовій DFD-діграмі закладу харчування

ІСНУЮЧІ РІШЕННЯ

	Ціна, $/ місяць	Автоматизація	Зворотній зв'язок	Чат	Статистика
Zendesk	109	Відсутні	–	+	+
Intercom	159	Відсутні	–	+	–
Google Places	0	Відсутні	–	–	–

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити веб-додаток, що стане допоміжним інструментом для моніторингу скарг у сфері послуг.

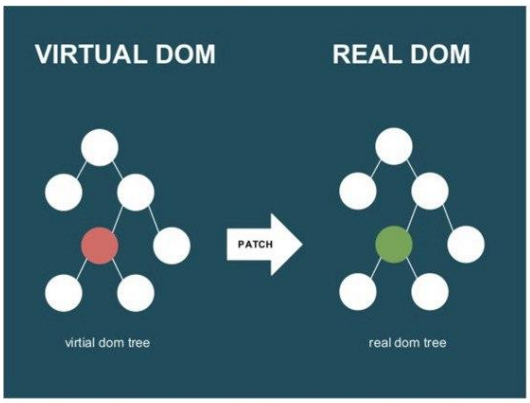
Завдання:

- Проаналізувати переваги та недоліки існуючих рішень;
- Визначити функціональні та нефункціональні вимоги до системи;
- Реалізувати веб-додаток;
- Провести тестування розробленого ПЗ;
- Визначити шляхи подальшого вдосконалення проєкту.

ЗАСОБИ РОЗРОБЛЕННЯ. КЛІЄНТСЬКА ЧАСТИНА

ReactJS

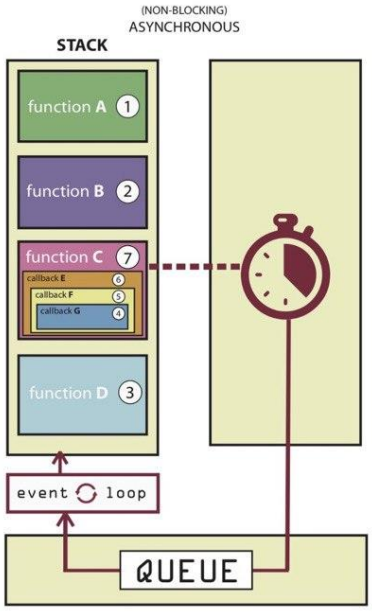
- Підтримка TypeScript;
- Оптимізовані механізми для динамічного оновлення сторінок;
- Найбільша спільнота розробників;
- Є бібліотекою, а не фреймворком.



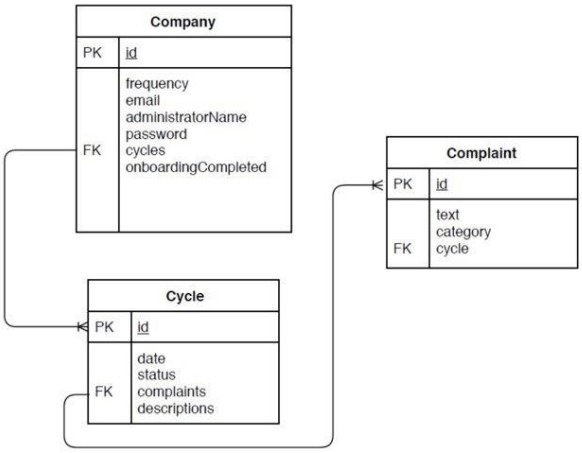
ЗАСОБИ РОЗРОБЛЕННЯ. СЕРВЕРНА ЧАСТИНА

NestJS

- Підтримка TypeScript;
- Асинхронна обробка запитів на сервер;
- Наявність декораторів, валідаторів, сокетів за замовчуванням.



ЗАСОБИ РОЗРОБЛЕННЯ. БАЗА ДАНИХ



MongoDB

- Документна БД;
- Зручна для прототипування;
- Індексування за будь-яким атрибутом;
- Можливість автоматичного шардингу;
- Широкий арсенал запитів.

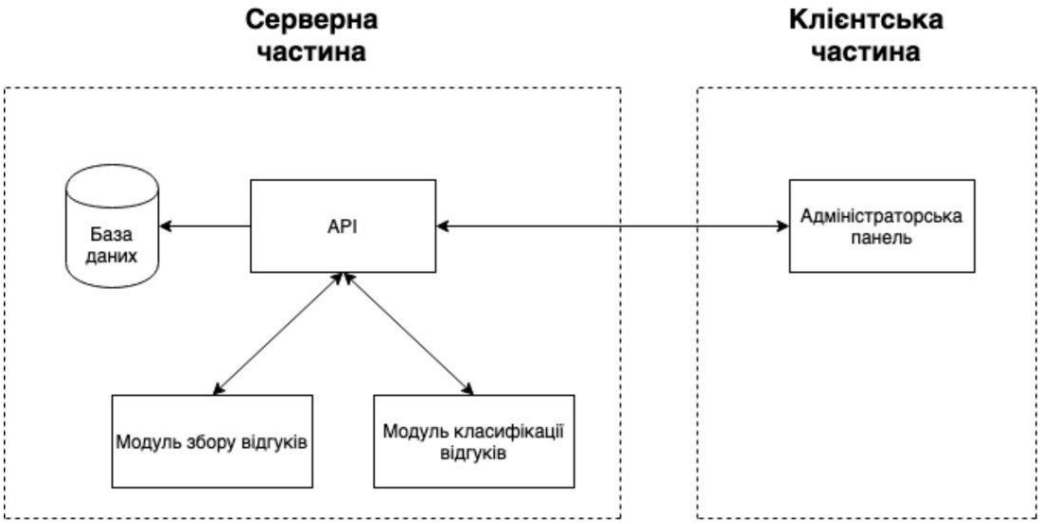
ЗАСОБИ РОЗРОБЛЕННЯ. ВИБІР МОДЕЛІ ДЛЯ КЛАСИФІКАЦІЇ

Порівняння моделей

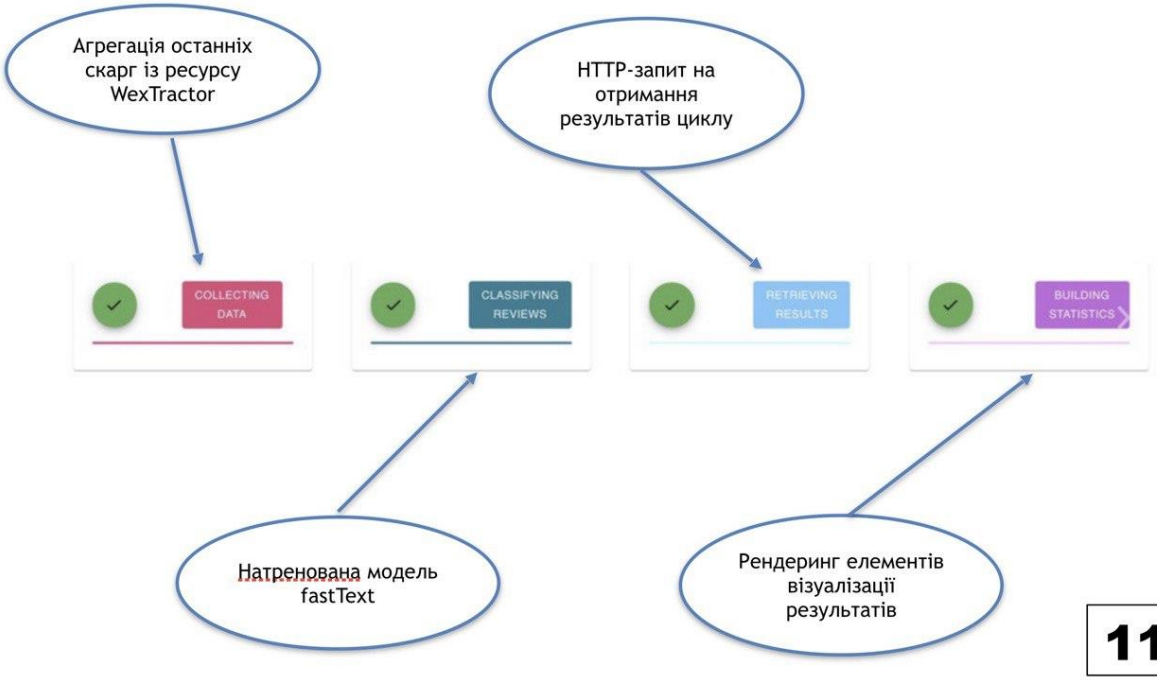
Модель	Точність, %	Час виконання, хв.
fastText	89.46	16.0
TextCNN	88.57	17.2
TextRNN	88.07	21.5
CharCNN	87.70	13.08
Transformer	88.54	46.47

- 14Гб ОЗУ;
- Intel Core i5-10400 2.9GHz;
- NVIDIA Tesla K80 GPU.

УЗАГАЛЬНЕНА ІНФРАСТРУКТУРА РОЗРОБЛЕНОГО ПЗ



ПЕРЕБІГ ВИКОНАННЯ ЦИКЛУ ЗБОРУ ТА АНАЛІЗУ СКАРГ



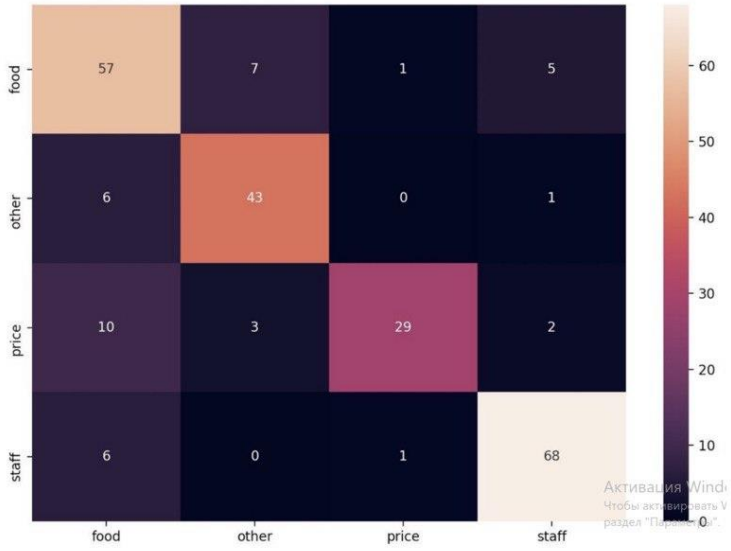
АНАЛІЗ ЕФЕКТИВНОСТІ КЛАСИФІКАЦІЇ ВИЯВЛЕНИХ СКАРГ

	Precision	Recall	F1-score	Support
food	0.72	0.81	0.77	70
other	0.81	0.86	0.83	50
price	0.94	0.66	0.77	44
staff	0.89	0.91	0.90	75

- Precision — вказує, яка частка ідентифікацій насправді була правильною;
- Recall — вказує, яка частка фактичних результатів була визначена правильно;
- F1-score — краща міра, якщо існує нерівномірний розподіл класів (велика кількість фактичних негативів).

АНАЛІЗ ЕФЕКТИВНОСТІ МОДЕЛІ

Матриця хибності



ТЕСТУВАННЯ ПРАЦЕЗДАТНОСТІ РОЗРОБЛЕНОЇ ІНФРАСТРУКТУРИ

АПРОБАЦІЯ

1. Опубліковані тези з теми: “System for monitoring online complaints through public API and NLP Techniques” у рамках конференції Intellectual Systems of Decision-Making and Problems of Computational Intelligence,International Scientific Conference.

2. Підготовлено пакет документів на отримання авторського свідоцтва.

ВИСНОВКИ

У ході реалізації дипломного проєкту були виконані наступні етапи проектування й розробки:

- Проведений збір функціональних та нефункціональних вимог до ПЗ;
- Проаналізовані переваги й недоліки існуючих рішень: Zendesk, Intercom, Google Places;
- Розроблений веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів;
- Проведене мануальне тестування за наперед визначеними тестовими випадками.

Дякую за увагу!

Додаток 3

Лістинг

3.1. Клієнтський компонент Pipeline

```
import React, {useEffect} from 'react';
import {Badge, Card, CardContent, CardHeader, createStyles, Divider,
Theme, Typography} from "@material-ui/core";
import {Progress} from "../Graphics/Progress";
import {useAppInjection} from "../../core/inversify";
import {EVENT_TYPES} from "../../services/Socket";
import {observer} from "mobx-react";
import Button from "@material-ui/core/Button";
import clsx from "clsx";
import {makeStyles} from "@material-ui/core/styles";
import {green} from "@material-ui/core/colors";

const useStyles = makeStyles((theme: Theme) =>
  createStyles({
    buttonSuccess: {
      height: 80,
      width: 200,
      justifyContent: 'center',
      alignItems: 'center',
      alignSelf: 'center',
      backgroundColor: green[500],
      '&:hover': {
        backgroundColor: green[700],
      },
    },
  }),
);

export const Pipeline = observer(() => {
  const App = useAppInjection();
  const { state } = App.companyController.socketService;
  const classes = useStyles();

  const buttonClassname = clsx({
    [classes.buttonSuccess]: true,
  });
  return (
    <Card style={{ display: 'flex', flex: 1, marginBottom: 50 }}>
      <div>
        <Badge color="secondary"
badgeContent={App.companyController.socketService.state} style={{
marginTop : 50 }}>
          <CardHeader style={{ paddingLeft: 100, }} title="Pipeline"
/>
        </Badge>
      </div>
      <div style={{ display: 'flex', flex: 1, justifyContent: 'flex-
start', paddingLeft: 175 }}>
```

```
        <Divider />
        <CardContent>
          <div style={{ display: 'flex', flex: 2, flexDirection:
'row', justifyContent: 'center'}}>
            <Card style={{ width: 300, marginRight: 50, }}>
              <CardContent>
                <Progress
loading={App.companyController.socketService.state !==
EVENT_TYPES.IDLE} finished={state !== EVENT_TYPES.DATA_COLLECTION}
running={state === EVENT_TYPES.DATA_COLLECTION} color={'pink'}
spinnerColor={'pink'} label={'Collecting data'} />
              </CardContent>
            </Card>
            <Card style={{ width: 300, marginRight: 50 }}>
              <CardContent>
                <Progress
loading={App.companyController.socketService.state !==
EVENT_TYPES.IDLE} finished={state !== EVENT_TYPES.ANALYZING}
running={state === EVENT_TYPES.ANALYZING} color={'blue'}
spinnerColor={'blue'} label={'Classifying reviews'} />
              </CardContent>
            </Card>
            <Card style={{ width: 300, marginRight: 50 }}>
              <CardContent>
                <Progress
loading={App.companyController.socketService.state !==
EVENT_TYPES.IDLE} finished={state !==
EVENT_TYPES.RETRIEVING_RESULTS} running={state ===
EVENT_TYPES.RETRIEVING_RESULTS} color={'red'} spinnerColor={'red'}
label={'Retrieving results'} />
              </CardContent>
            </Card>
            <Card style={{ width: 300, marginRight: 50 }}>
              <CardContent>
                <Progress
loading={App.companyController.socketService.state !==
EVENT_TYPES.IDLE} finished={state !== EVENT_TYPES.PROCESSING}
running={state === EVENT_TYPES.PROCESSING} color={'orange'}
spinnerColor={'orange'} label={'Building statistics'} />
              </CardContent>
            </Card>
          </div>
        </CardContent>
        </div>
      </Card>
  )
});
```

3.2. Клієнтський контролер для роботи із компаніями

```
import { API_SERVICE_TYPE, IApiService } from "../services/Api";
import { inject, injectable } from "inversify";
import { COMPANY_REPOSITORY_TYPE, ICompanyRepository } from
"../repositories/CompanyRepository";
import { IRepository, REPOSITORY_TYPE } from
"../repositories/Repository";
import {AUTH_SERVICE_TYPE, IAuthService} from "../services/Auth";
import {ISocketService, SOCKET_SERVICE_TYPE} from
"../services/Socket";

export interface ICompanyController {
repository: ICompanyRepository;
authService: IAuthService;
socketService: ISocketService;
loginCompany(email: string, password: string, history: any): void;
completeOnBoarding: (administratorName: string, frequency: string,
company: string, history: any) => void;
registerCompany: (email: string, password: string, confirmPassword:
string, history: any) => Promise<void>;
getCompanyInfo: (id: string, token: string) => void;
logCompanyOut(history: any): void;
updateCompany(administratorName: string, frequency: string, email:
string): void;
}

export const COMPANY_CONTROLLER_TYPE = 'UserControllerType';

@injectable()
export class CompanyController implements ICompanyController {

public repository: ICompanyRepository;
private appRepository: IRepository;
private apiService: IApiService;
public authService: IAuthService;
public socketService: ISocketService;

constructor(
@inject(REPOSITORY_TYPE) appRepository: IRepository,
@inject(COMPANY_REPOSITORY_TYPE) repository: ICompanyRepository,
@inject(AUTH_SERVICE_TYPE) authService: IAuthService,
@inject(API_SERVICE_TYPE) apiService: IApiService,
@inject(SOCKET_SERVICE_TYPE) socketService: ISocketService,
) {
this.repository = repository;
this.appRepository = appRepository;
this.authService = authService;
this.apiService = apiService;
this.socketService = socketService;
this.socketService.emit('test');
}
```

```
public async loginCompany(email: string, password: string, history:
any) {
try {
const response = await this.apiService.login(email, password);
if (response.status !== 201) {
throw new Error();
}
const { access_token, id } = response.data;
await this.getCompanyInfo(id, access_token);
await this.authService.login(access_token, id);
if (this.repository.company.onboarding_completed) {
history.push('/dashboard');
} else {
history.push('/onboarding');
}
} catch (e) {
console.log(e);
}
}

public async getCompanyInfo(id: string, token: string) {
try {
const response = await this.apiService.getCompany(id, token!);
this.repository.company = {...this.repository.company,
...response.data, id, token};
} catch(e) {
console.log(e);
}
}

public async registerCompany(email: string, password: string,
confirmPassword: string, history: any) {
try {
const {data, status} = await this.apiService.register(email,
password);
if (status !== 201) {
throw new Error();
}
this.repository.readyToLogin = true;
} catch(e) {
console.log(e);
}
}

public async completeOnBoarding(administratorName: string,
frequency: string, company: string, history: any) {
try {
console.log(this.repository.company.id,
this.repository.company.token);
```

```
const { data, status } = await
this.apiService.completeOnBoarding(administratorName, frequency,
company, this.repository.company.id,
this.repository.company.token!);
if (status !== 200) {
throw new Error();
}
history.push('/dashboard')
}
catch (e) {
console.log(e);
}
}

public async updateCompany(administratorName: string, frequency:
string, email: string) {
try {
const {id, token} = this.repository.company;
const { status, data } = this.apiService.updateCompany(id, token,
{administratorName, frequency, email});
if (status === 200) {
this.repository.company = data;
this.repository.setSession(id, token!);
}
} catch (e) {
console.log(e);
}
}

public async logCompanyOut(history: any) {
try {
this.authService.logout();
history.push('/');
} catch(e) {

}
}
}
```

3.3. Серверний контролер для обробки запитів для компаній

```
import { Body, Controller, Post, Put, Req, UseGuards, Get, Param }
from '@nestjs/common';
import { CompleteOnBoardingDto, LoginDto, RegisterCompanyDto,
UpdateCompanyDto } from '../decorators/validators/companies.dto';
import { AuthService } from '../auth/auth.service';
import { CompaniesService } from './companies.service';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';

@Controller('companies')
export class CompaniesController {

  constructor(
    private authService: AuthService,
    private companiesService: CompaniesService,
  ) {}

  @UseGuards(JwtAuthGuard)
  @Get(':id')
  async getCompany(@Param('id')id: any) {
    return {
      data: {
        ...(await this.companiesService.findById(id))._doc
      }
    };
  }

  @Post('register')
  registerCompany(@Body() registerCompanyDto: RegisterCompanyDto) {
    return this.authService.register(registerCompanyDto);
  }

  @Post('login')
  login(@Body() loginDto: LoginDto) {
    return this.authService.login(loginDto);
  }

  @UseGuards(JwtAuthGuard)
  @Put('complete-onboarding')
  completeOnBoarding(@Body() completeOnBoardingDto:
CompleteOnBoardingDto) {
    return this.companiesService.update({...completeOnBoardingDto,
onboarding_completed: true});
  }

  @UseGuards(JwtAuthGuard)
  @Put('update')
  updateCompanyInfo(@Body() updateCompanyDto: UpdateCompanyDto) {
    return this.companiesService.update(updateCompanyDto);
  }
}
```

3.4. Клієнтський контролер для роботи із циклами

```
import { inject, injectable } from "inversify";
import {CYCLE_REPOSITORY_TYPE, IComplaint, ICycleRepository} from
"../repositories/CycleRepository";
import {API_SERVICE_TYPE, IApiService} from "../services/Api";

export interface ICycleController {
  repository: ICycleRepository;
  getCycles: () => Promise<void>;
  addCycle: (cycleData: any) => Promise<void>;
  startCycle(id: string): void;
  getComplaints(): void;
  getPrevCycleComplaints(): void;
}

export const CYCLE_STATUSES = {
  CREATED: 'created',
  PENDING: 'pending',
  COMPLETED: 'completed',
  ABORTED: 'aborted',
};
export const CYCLE_CONTROLLER_TYPE = 'CYCLE_CONTROLLER_TYPE';
@injectable()
export class CycleController implements ICycleController {

  public repository: ICycleRepository;
  private apiService: IApiService;

  constructor(
    @inject(CYCLE_REPOSITORY_TYPE) repository: ICycleRepository,
    @inject(API_SERVICE_TYPE) apiService: IApiService,
  ) {
    this.repository = repository;
    this.apiService = apiService;
  }

  async addCycle(cycleData: any) {
    try {
      const {status, data} = await this.apiService.addCycle({
...cycleData, companyId: this.repository.session.id, date: new
Date().toString() }, this.repository.session.token);
      if (status !== 200) {
        throw new Error()
      }
      await this.getCycles();
    } catch(e) {
      console.log(e);
    }
  }

  async startCycle(id: string) {
```

```
    try {
      const { status, data } = await this.apiService.startCycle(id,
this.repository.session.token);
      if (status !== 200) {
        throw new Error();
      }
    } catch(e) {
      console.log(e);
    }
  }

  async getCycles() {
    try {
      const {id, token} = this.repository.session;
      const {data, status} = await this.apiService.getCycles(id,
token);
      if (status !== 200) {
        throw new Error();
      }

      this.repository.cycles = data.cycles.map((cycle: any) =>
({...cycle, id: cycle._id}));
      const completedCycles = data.cycles.filter(item => item.status
=== 'completed');
      if (completedCycles === 0) {
        return;
      }
      const latestCycle = completedCycles.length > 0 ?
completedCycles[completedCycles.length - 1] : null;
      this.repository.latestCycle = {...latestCycle, id:
latestCycle._id, issues: { food: [], staff: [], other: [], prices:
[] } };

      if (completedCycles.length < 2) {
        return;
      }
      const prevCycle = completedCycles.length > 1 ?
completedCycles[data.cycles.length - 2] : null;
      this.repository.prevCycle = {...prevCycle, id: prevCycle._id,
issues: { food: [], staff: [], other: [], prices: [] } };
    }
     catch(e) {
      console.log(e);
    }
  }

  async getComplaints() {
    try {
      const { status, data } = await
this.apiService.getComplaints(this.repository.latestCycle.id,
this.repository.session.token);
```

```
      if (status !== 200 || !data) {
        throw new Error();
      }
      this.repository.latestCycle.issues = {
        food: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_food').map(complaint => ({...complaint, category:
'Food'})),
        prices: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_price').map(complaint => ({...complaint, category:
'Prices'})),
        staff: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_staff').map(complaint => ({...complaint, category:
'Staff'})),
        other: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_other').map(complaint => ({...complaint, category:
'Other'})),
      };
    } catch(e) {
      console.log(e);
    }
  }

  async getPrevCycleComplaints() {
    try {
      if (!this.repository.prevCycle) {
        return;
      }
      const { status, data } = await
this.apiService.getComplaints(this.repository.prevCycle.id,
this.repository.session.token);
      if (status !== 200 || !data) {
        throw new Error();
      }
      this.repository.prevCycle.issues = {
        food: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_food').map(complaint => ({...complaint, category:
'Food'})),
        prices: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_price').map(complaint => ({...complaint, category:
'Prices'})),
        staff: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_staff').map(complaint => ({...complaint, category:
'Staff'})),
```

```
        other: (Object.values(data) as
IComplaint[]).filter((complaint: IComplaint) => complaint.category
=== 'negative_other').map(complaint => ({...complaint, category:
'Other'})),
      };
    } catch(e) {
      console.log(e);
    }
  }
}
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗОВАННОГО ВИЯЛЕННЯ ТА ОБРОБЛЕННЯ СКАРГ ІНТЕРНЕТ-КОРИСТУВАЧІВ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ГЕРАСИМОВ

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-додаток для автоматизованого виявлення та оброблення скарг Інтернет-користувачів. Дана програмна система реалізована у вигляді комплексу веб-додатка, імплементованого за допомогою фреймворку React, та серверних сервісів. API розроблений із використанням фреймворку Nest.js, а алгоритми машинного навчання впроваджені із використання мови програмування Python та типових бібліотек.

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних елементів:

1. Відповідність розробленого ПЗ вимогам, висунутим при проєктування.
2. Реєстрація компанії та заповнення початкових даних через веб-додаток.
3. Створення циклу збору та аналізу даних із специфікацією додаткової інформації,
4. Перегляд поточного стану виконання циклу збору та аналізу даних у реальному часі,
5. Перевірка функціонування бази даних.
6. Перегляд результатів аналізу в адміністраторській панелі.
7. Забезпечення безпеки даних.
8. Інтуїтивність інтерфейсу користувача.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування даної системи передбачає повний комплекс заходів по Unit-тестуванню, інтеграційним перевіркам та end-to-end переглядів для автоматизованого виявлення локальних та глобальних проблем компонентів програмної системи. Для Unit-тестування клієнтської частини рекомендується фреймворк Jest із інструментом Enzyme. Дані бібліотеки

дозволять тестувати граничні випадки використання системи та перевіряти коректність рендерингу компонентів на етапі їх приєднання до DOM-дерева. Інтеграційні тести рекомендовано виконувати із застосування інструменту CucumberJS, що єдиний у своєму класі має підтримку Typescript. Даний інструмент дозволить протестувати серверну частину застосунку абстрагуючись від сценаріїв клієнтського досвіду. Таке тестування виявляє потенційні проблему серверу на етапі масштабування. Розповсюдженим для end-to-end тестування веб-додатків є фреймвор Cypress, що дозволяє імітувати взаємодію користувача із адміністраторською панеллю у режимі реального часу із ілюстрацією кожного кроку виконання. Результати усіх перевірок рекомендується зберігати у вигляді Allure звіту для більш наглядного представлення для розробників.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Повний процес тестування проходить у такому порядку:

1. Unit-тестування окремих компонентів системи.
2. Інтеграційне тестування серверних сервісів.
3. Автоматизоване тестування веб-додатку.
4. Тестування під навантаженням.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗОВАННОГО ВИЯЛЕННЯ ТА ОБРОБЛЕННЯ СКАРГ ІНТЕРНЕТ-КОРИСТУВАЧІВ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Артем ГЕРАСИМОВ

2020

ЗМІСТ

1. ОПИС СТРУКТУРИ WEB-РЕСУРСУ

Програмна система реалізована у вигляді веб-додатку, що складається виключно із динамічно-оновлюваних сторінок.

Інтерфейс користувача є одномовним, використовується виключно англійська мова. Навігація системою є інтуїтивною та відкритою, окрім обмеження доступу до певних сторінок для неавторизованих користувачів.

У системі наявні такі динамічні веб-сторінки:

1. Сторінка реєстрації/авторизації.
2. Сторінка заповнення початкових даних про компанію.
3. Сторінка налаштувань компанії.
4. Сторінка із архівом циклів та формою створення нових циклів.
5. Сторінка відображення результатів досліджень

Перемикання між трьома основними сторінками системи відбувається за допомогою навігаційної панелі у верхній частині сторінок. Так, є можливість швидкого перемикання між сторінками налаштувань, архівом циклів та статистичною панеллю.

Якщо користувач не є авторизованим у системі, він має доступ лише до сторінки реєстрації/авторизації. Під час першої авторизації, користувачу пропонується заповнити початкові дані про компанію. Подальше використання системи є неможливим без проходження даного етапу.

2. ВМІСТ ДИНАМІЧНИХ WEB-СТОРІНОК

Сторінка із результатами аналізу скарг містить панель із поточним статусом виконання (рис. 2.1). Статуси динамічно оновлюється, а у користувача є можливість зупинити процес виконання на будь-якому етапі.

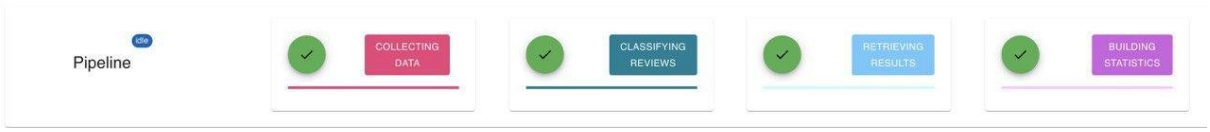


Рис. 2.1. Панель із поточними статусами виконання

Також, статистична сторінка містить результати останнього циклу збору та аналізу скарг. Користувач має можливість переглянути категорії скарг, скарги по кожній категорії та список усіх виявлених скарг (рис. 2.2).

Рис. 2.2. Панель із результатами останнього циклу аналізу

Адміністратор будь-якої компанії має можливість зареєструватися та авторизуватися у системі через сторінку реєстрації/авторизації (рис.2.3).

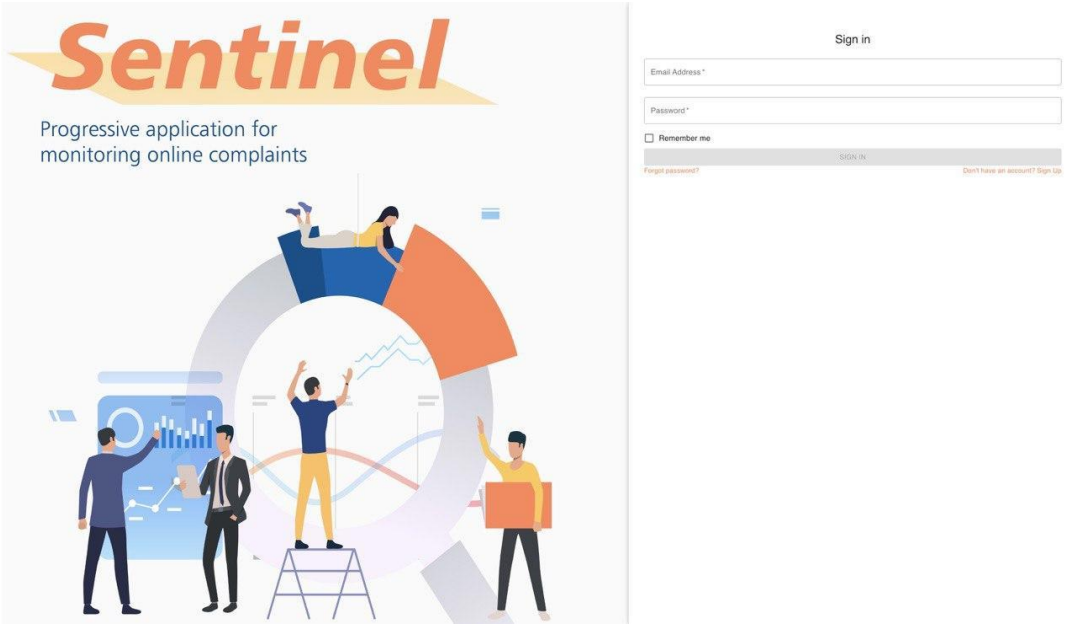


Рис. 2.3. Сторінка авторизації/реєстрації

3. ПРОЦЕДУРА РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ

Реєстрація нової компанії відбувається на початковій сторінці, для цього необхідно натиснути на кнопку “Don’t have an account? Sign Up”. Форма авторизації динамічно оновиться й користувачу буде запропоновано ввести поштову адресу та пароль (рис. 3.1).

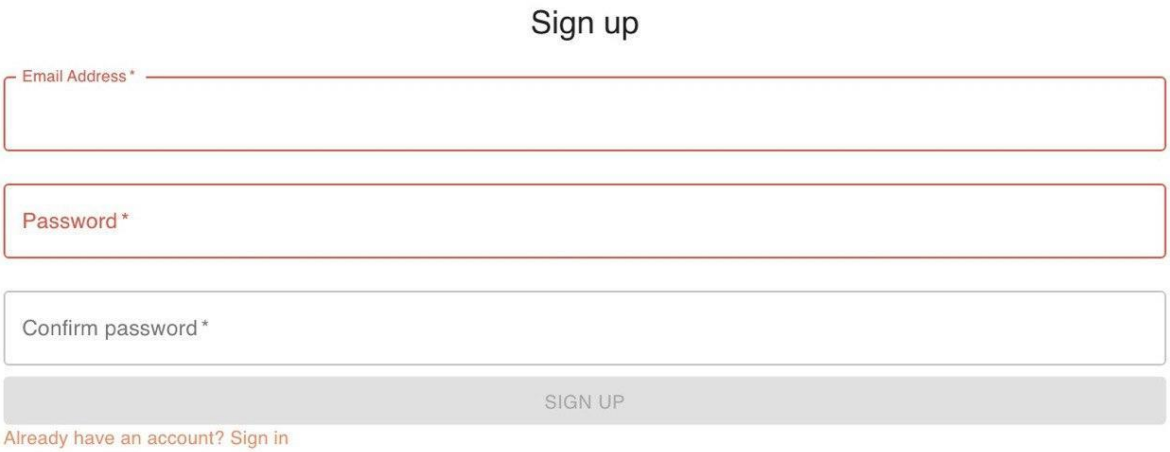


Рис. 3.1. Форма реєстрації

Після проходження валідації полей та успішної реєстрації форма знову оновиться й користувачу за пропонується ввести поштову адресу й пароль для авторизації (рис. 3.2).

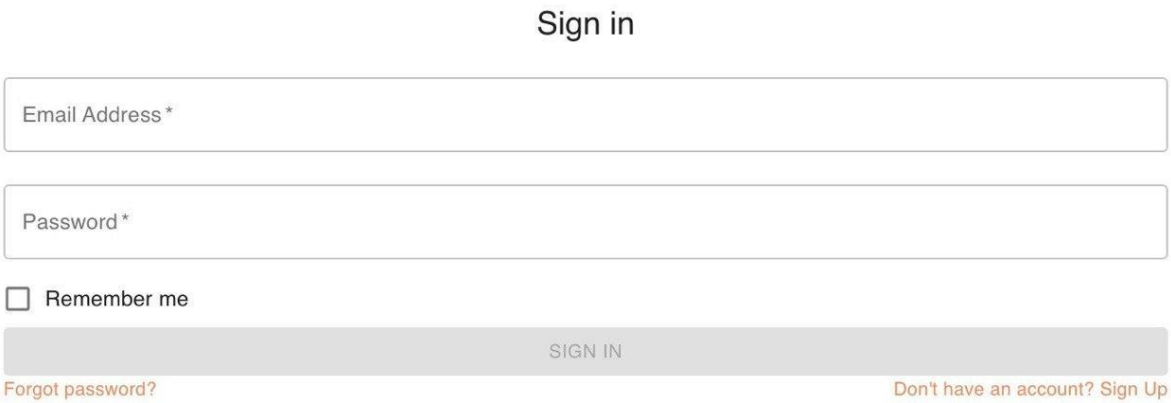


Рис. 3.2. Форма авторизації

Після першої авторизації користувачу пропонується заповнити початкові дані про компанію та вказати частоту проведення циклів збору та аналізу (рис. 3.3).

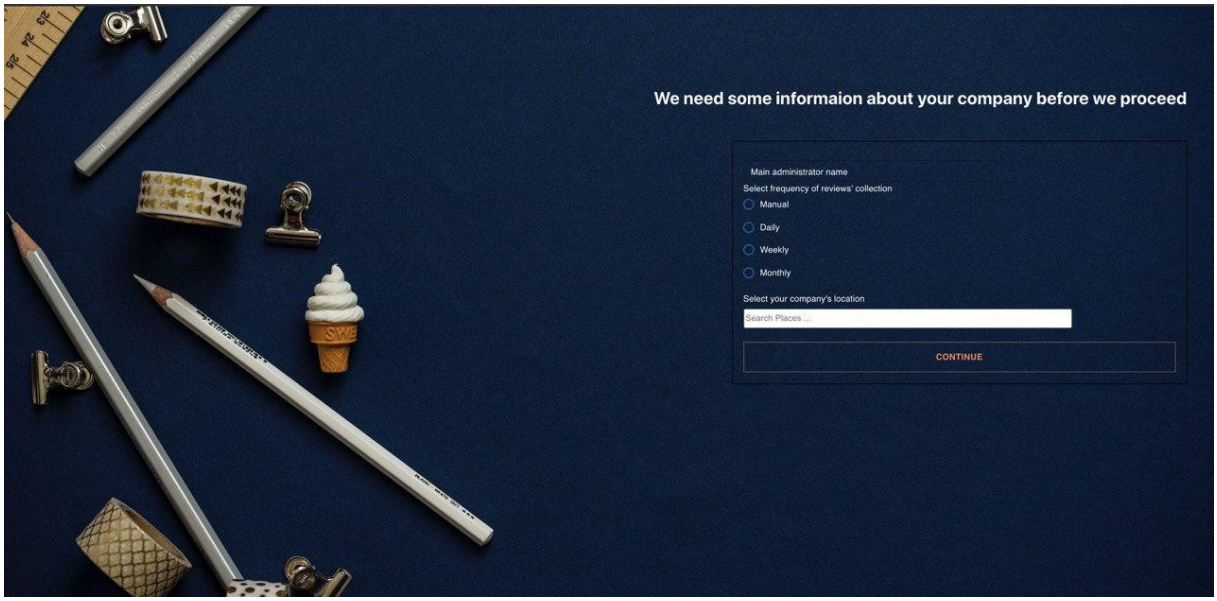


Рис. 3.3. Сторінка заповнення початкових даних

4. АДМІНІСТРУВАННЯ СИСТЕМИ

Для ефективного функціонування системи адміністратору закладу необхідно виконувати дії для запуску, спостереження та аналізу результатів проведених циклів. Першим кроком є створення циклу у вкладці “Cycles”, де для відображення форми створення необхідно натиснути кнопку “Add cycle”. Після специфікації параметрів циклу, необхідно натиснути кнопку “Add cycle” (рис. 4.1) та запустити цикл у загальному списку (рис. 4.2).

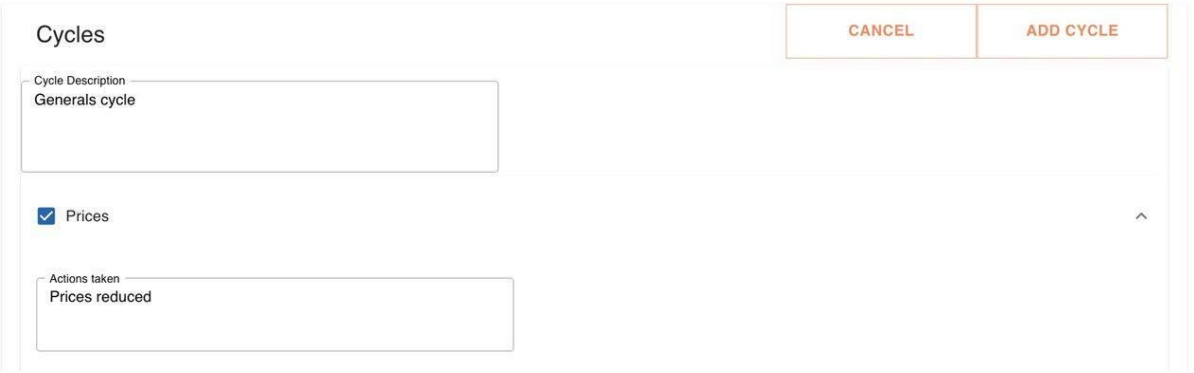


Рис. 4.1. Форма створення нового циклу збору та аналізу скарг

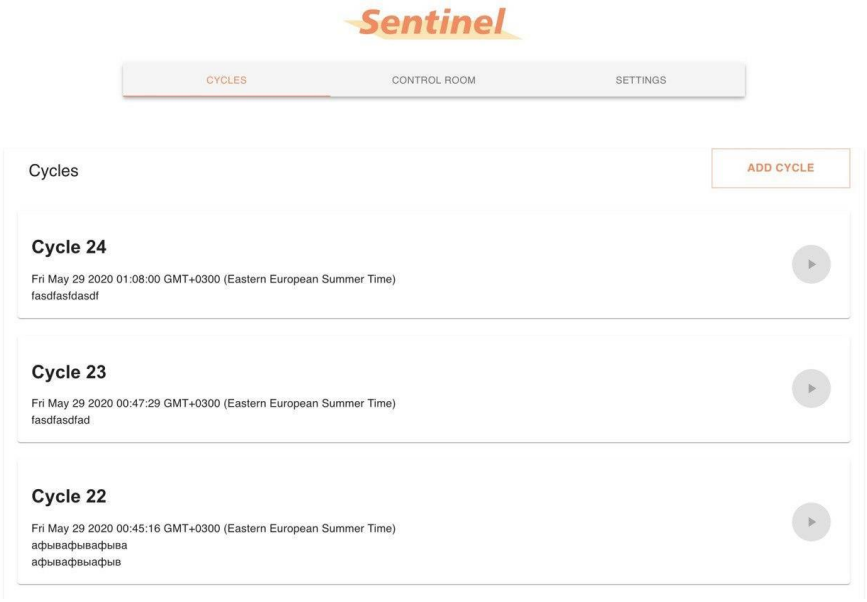


Рис. 4.2. Загальний список створених циклів

Для перегляду результатів виконання циклу необхідно перейти у вкладку “Control Room”, де можна побачити діаграми проблем та списки проблем із відповідними скаргами (рис. 4.3).

Рис. 4.3. Сторінка результатів останнього циклу збору та аналізу скарг

Список представляє скорочені тексти скарг, тому для перегляду повної версії певного коментаря необхідно натиснути на нього для виклику модального вікна з деталями (рис. 4.4).

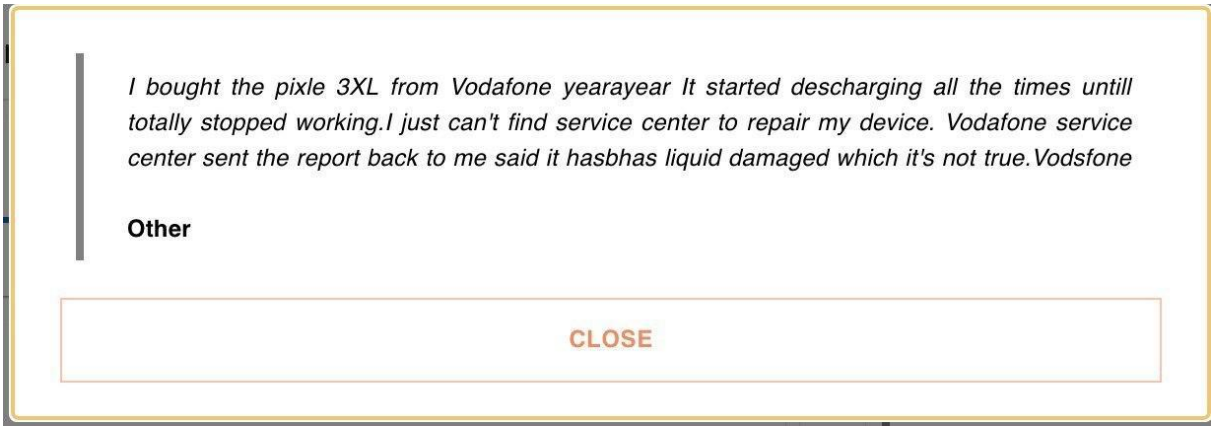


Рис. 4.4. Модальне вікно із деталями скарги